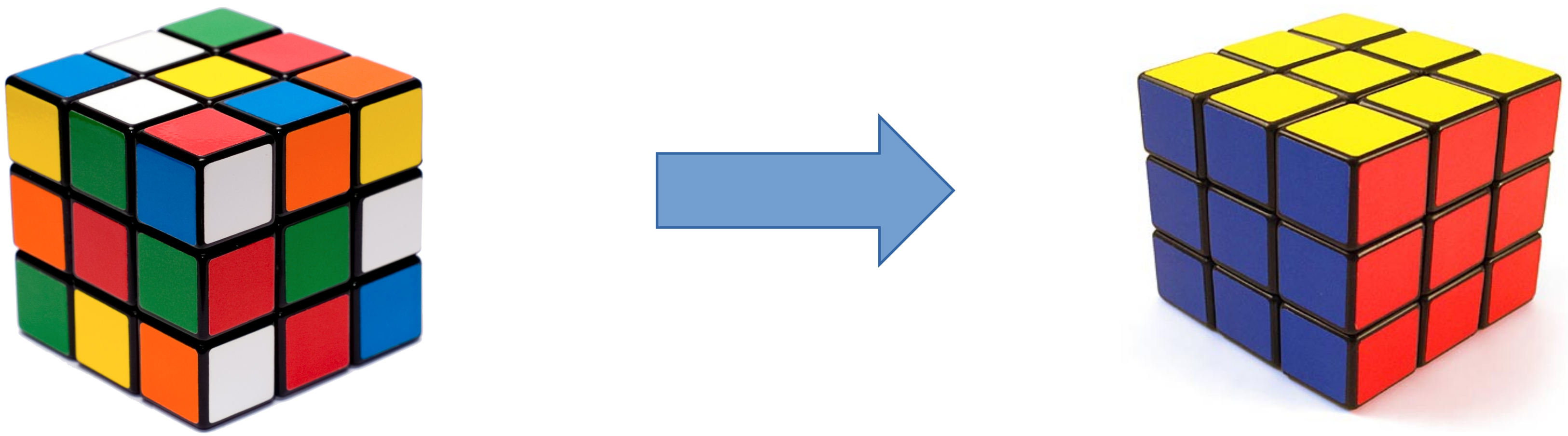
Review

Chris Amato
Northeastern University

Some images and slides are used from: Rob Platt, CS188 UC Berkeley, AIMA

Search

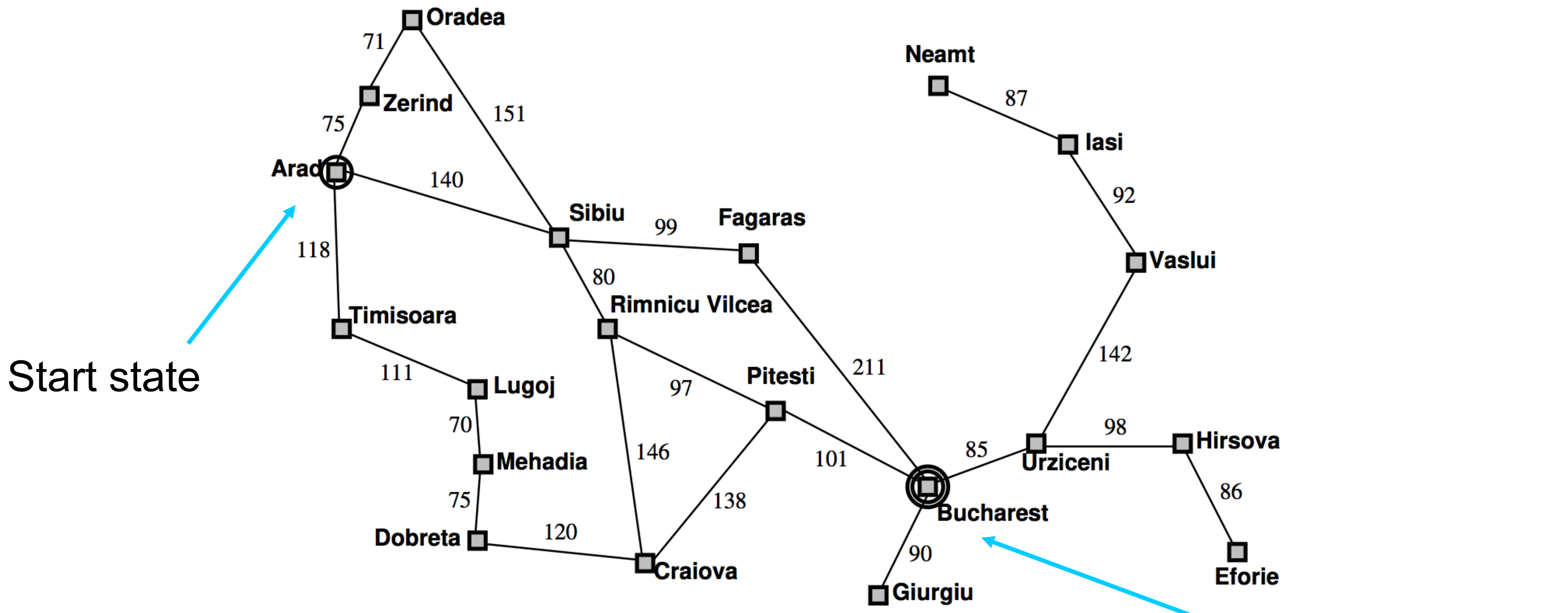
What is graph search?



Graph search: find a path from start to goal

- what are the states?
- what are the actions (transitions)?
- how is this a graph?

What is graph search?



Graph search: find a path from start to goal

– what are the states?

– what are the actions (transitions)?

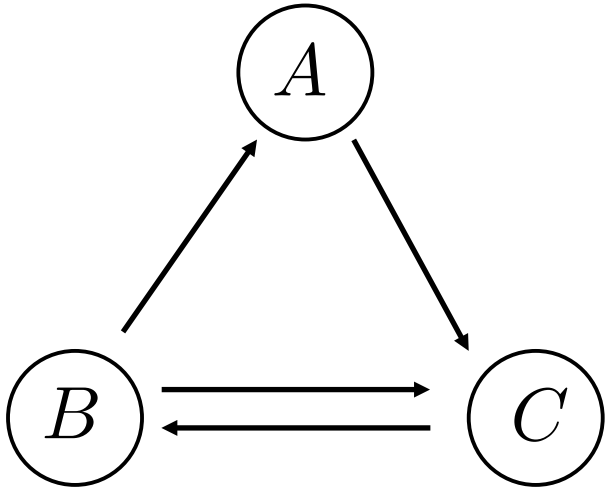
– how is this a graph?

What is a graph?

Graph: $G = (V, E)$

Vertices: V

Edges: E



Directed graph

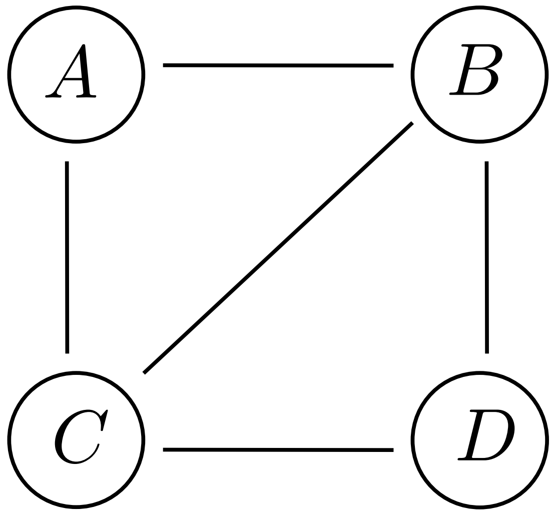
$$V = \{A, B, C\}$$
$$E = \{(B, A), (A, C), (B, C), (C, B)\}$$

What is a graph?

Graph: $G = (V, E)$

Vertices: V

Edges: E

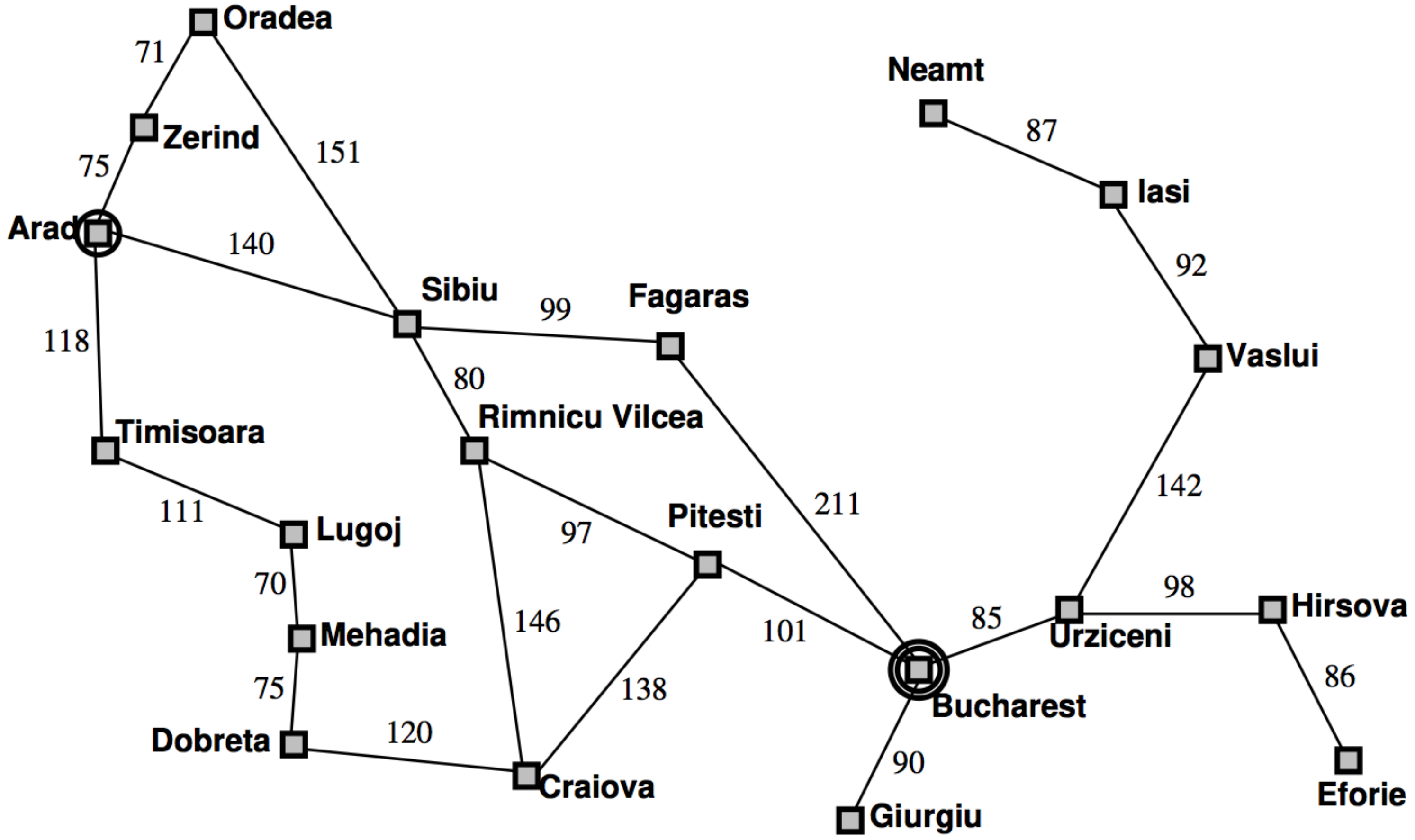


Undirected graph

$$V = \{A, B, C, D\}$$

$$E = \{\{A, C\}, \{A, B\}, \{C, D\}, \{B, D\}, \{C, B\}\}$$

Graph search



Given: a graph, *G*

Problem: find a path from A to B

– A: start state

– B: goal state

Problem formulation

A problem is defined by four items:

– *initial state* e.g., “at Arad”

– *successor function* S(x) = set of action–state pairs

e.g., S(Arad) = {⟨Arad → Zerind, Zerind⟩, . . .}

– *goal test*

can be explicit, e.g., x = “at Bucharest” implicit, e.g., NoDirt(x)

– *path cost* (additive)

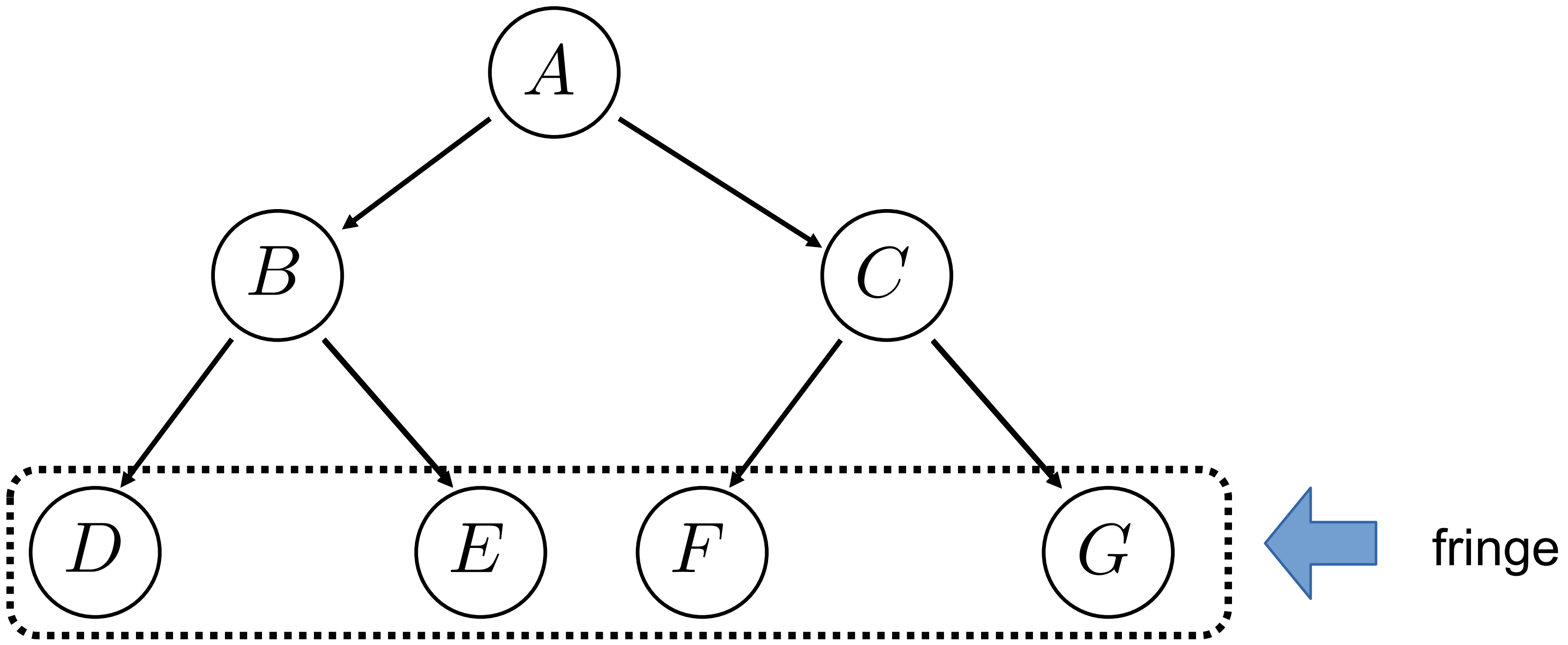
e.g., sum of distances, number of actions executed, etc.

c(x, a, y) is the step cost, assumed to be ≥ 0

A solution is a sequence of actions leading from the initial state to a goal state

Breadth first search (BFS)

Fringe
D
E
F
G

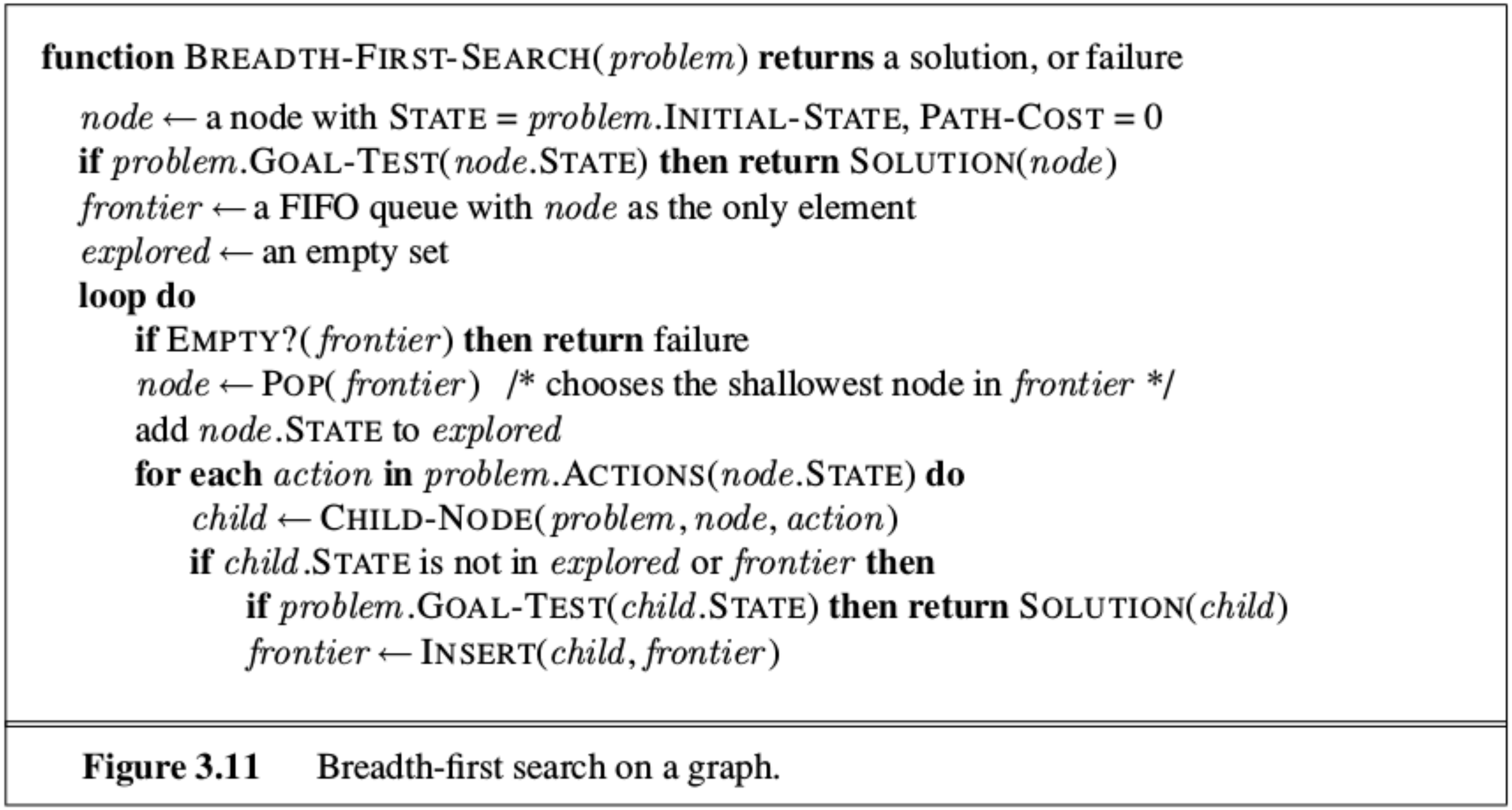


Which state gets removed next from the fringe?

What kind of a queue is this?

FIFO Queue!
(first in first out)

Breadth first search (BFS)



```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    frontier ← a FIFO queue with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the shallowest node in frontier */
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
                frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

BFS Properties

Is BFS complete?
– is it guaranteed to find a solution if one exists?

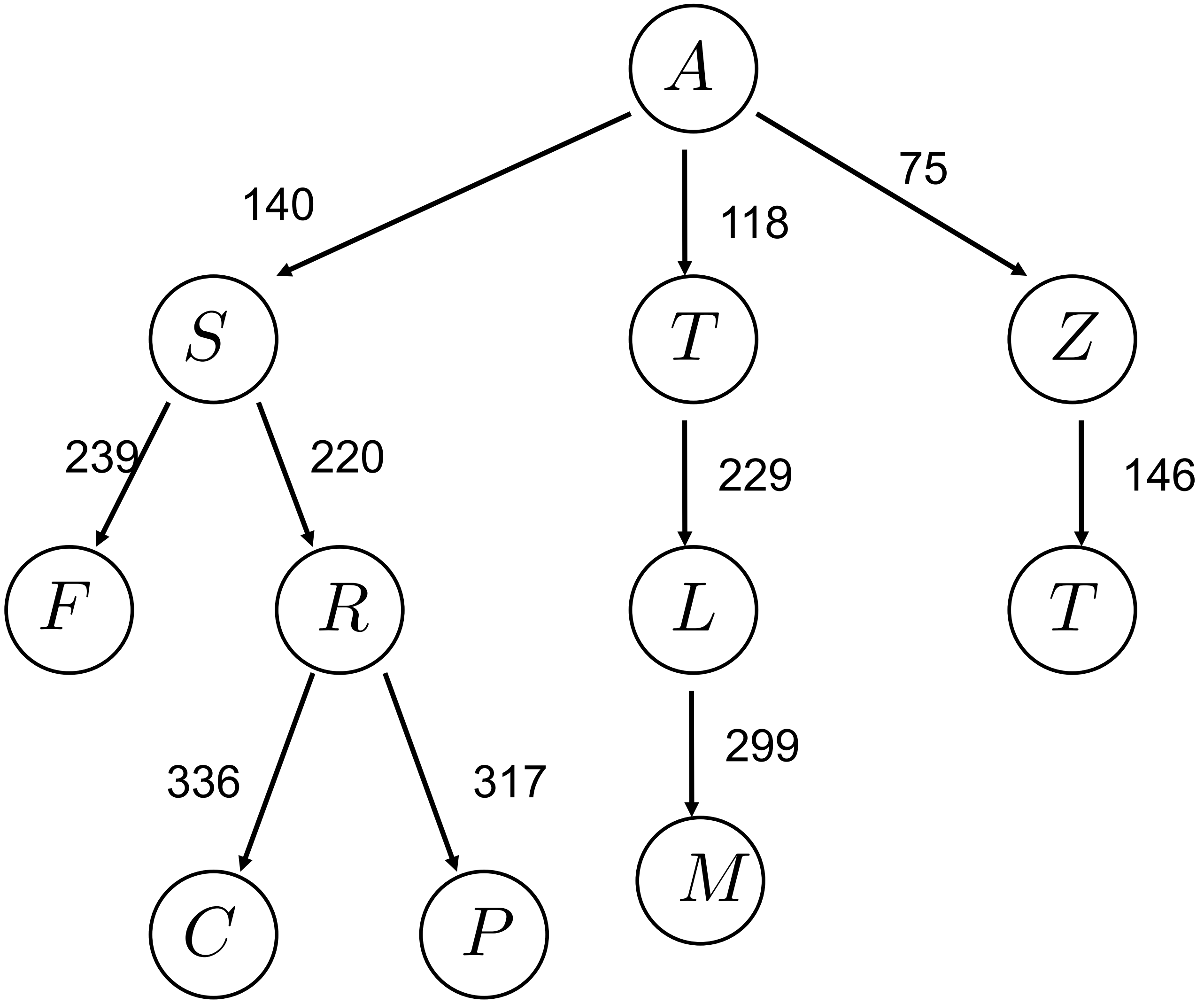
What is the time complexity of BFS?
– how many states are expanded before finding a solution?
- – b: branching factor
- – d: depth of shallowest solution
- – complexity = $O(b^d)$

What is the space complexity of BFS?
– how much memory is required?
- – complexity = $O(b^d)$

Is BFS optimal?
– is it guaranteed to find the best solution (shortest path)?

UCS

Fringe	Path Cost
~~A~~	~~0~~
~~S~~	~~140~~
~~T~~	~~118~~
~~Z~~	~~75~~
~~T~~	~~146~~
~~L~~	~~229~~
F	239
~~R~~	~~220~~
C	336
P	317
M	299



Explored set: A, Z, T, S, R, L

UCS

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                frontier ← INSERT(child, frontier)
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
```

Figure 3.14 Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for *frontier* needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.

UCS Properties

Is UCS complete?

– is it guaranteed to find a solution if one exists?

What is the time complexity of UCS?

– how many states are expanded before finding a solution?

- – b: branching factor
- – C*: cost of optimal solution
- – e: min one-step cost
- – complexity = $O(b^{C^*/e})$

What is the space complexity of BFS?

– how much memory is required?

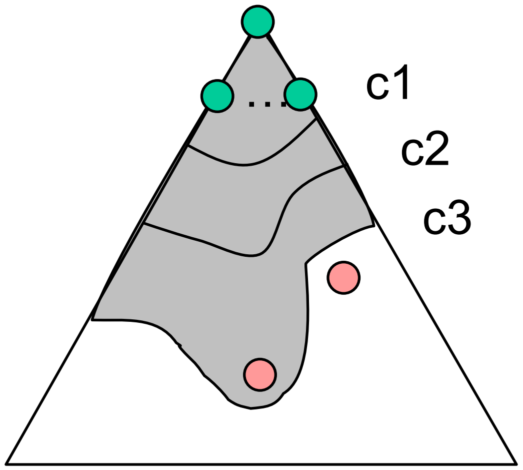
- – complexity = $O(b^{C^*/e})$

Is BFS optimal?

– is it guaranteed to find the best solution (shortest path)?

UCS vs BFS

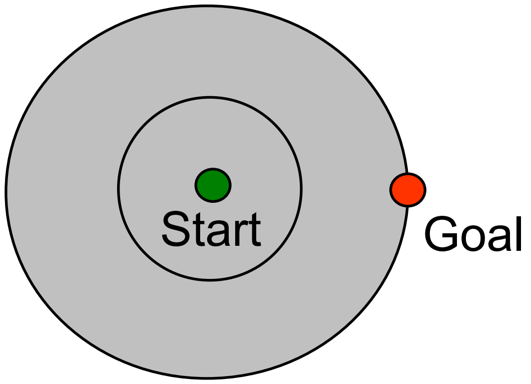
Remember: UCS explores increasing cost contours



The good: UCS is complete and optimal!

The bad:

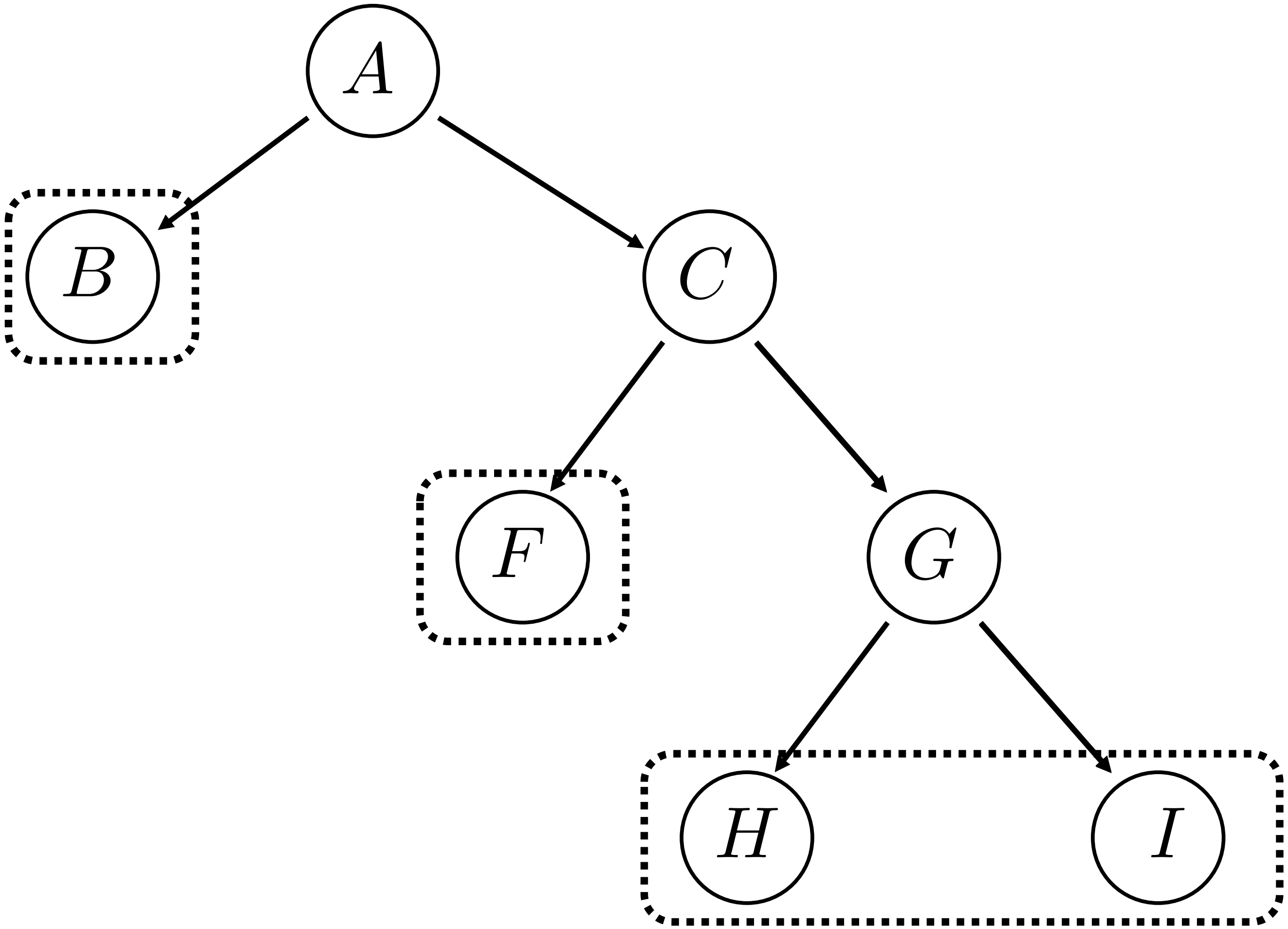
- Explores options in every “direction”
- No information about goal location



We’ll fix that soon!

DFS

Fringe
~~A~~
B
~~C~~
F
~~G~~
H
I



Which state gets removed next from the fringe?

What kind of a queue is this?

LIFO Queue!
(last in first out)

DFS Properties: Graph search version

This is the "graph search" version of the algorithm

Is DFS complete?
– only if you track the explored set in memory

What is the time complexity of DFS (graph version)?
– how many states are expanded before finding a solution?
    – complexity = number of states in the graph

What is the space complexity of DFS (graph version)?
– how much memory is required?
    – complexity = number of states in the graph

Is DFS optimal?
– is it guaranteed to find the best solution (shortest path)?

DFS: Tree search version

This is the “tree search” version of the algorithm

Suppose you <u>don't</u> track the explored set.
– why wouldn't you want to do that?

What is the <u>space complexity</u> of DFS (tree version)?
– how much memory is required?
- – b: branching factor
- – m: maximum depth of any node
- – complexity = $O(bm)$

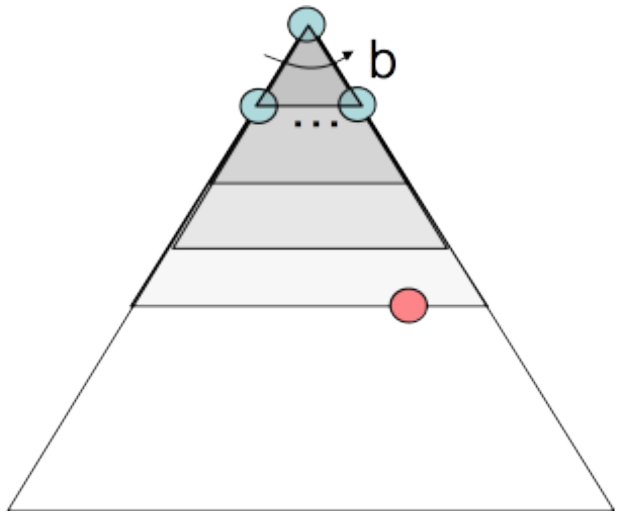
What is the <u>time complexity</u> of DFS (tree version)?
– how many states are expanded before finding a solution?
- – complexity = $O(b^m)$

Is it complete?

NO!
What do we do???

IDS: Iterative deepening search

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..



- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

IDS

What is the space complexity of IDS (tree version)?
– how much memory is required?
  – b: branching factor
  – m: maximum depth of any node
  – complexity = $O(bm)$

What is the time complexity of DFS (tree version)?
– how many states are expanded before finding a solution? $O(b^m)$
  – complexity =

Is it complete? YES!!!

Is it optimal? Sometimes!!!

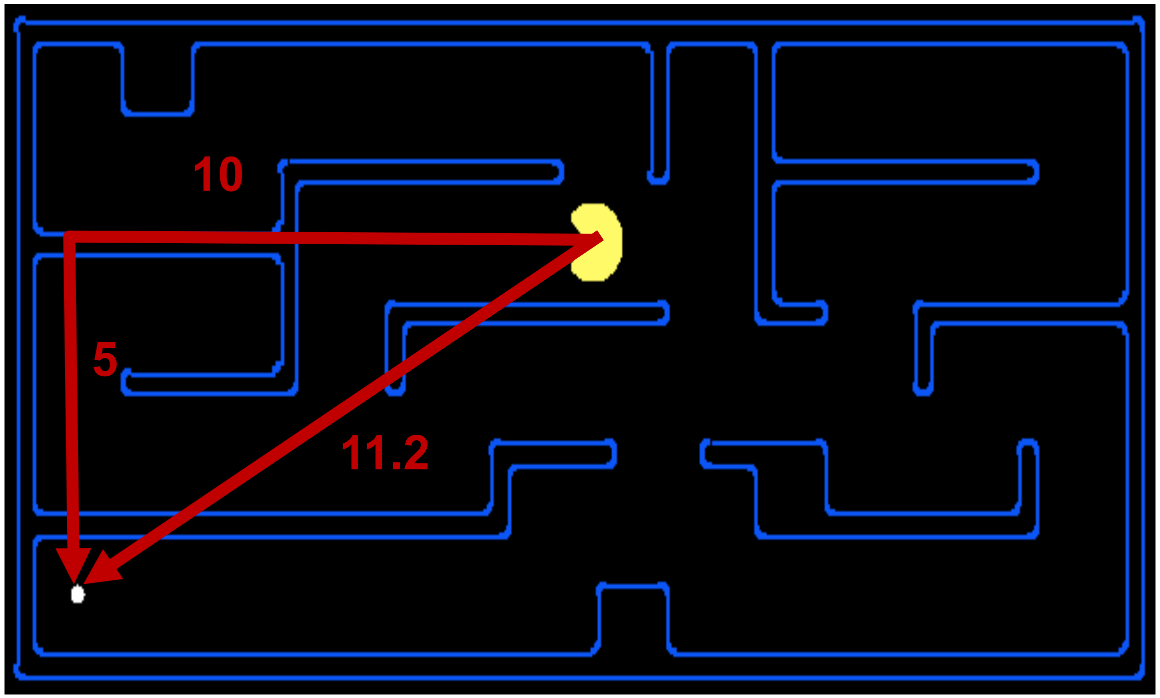
The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object

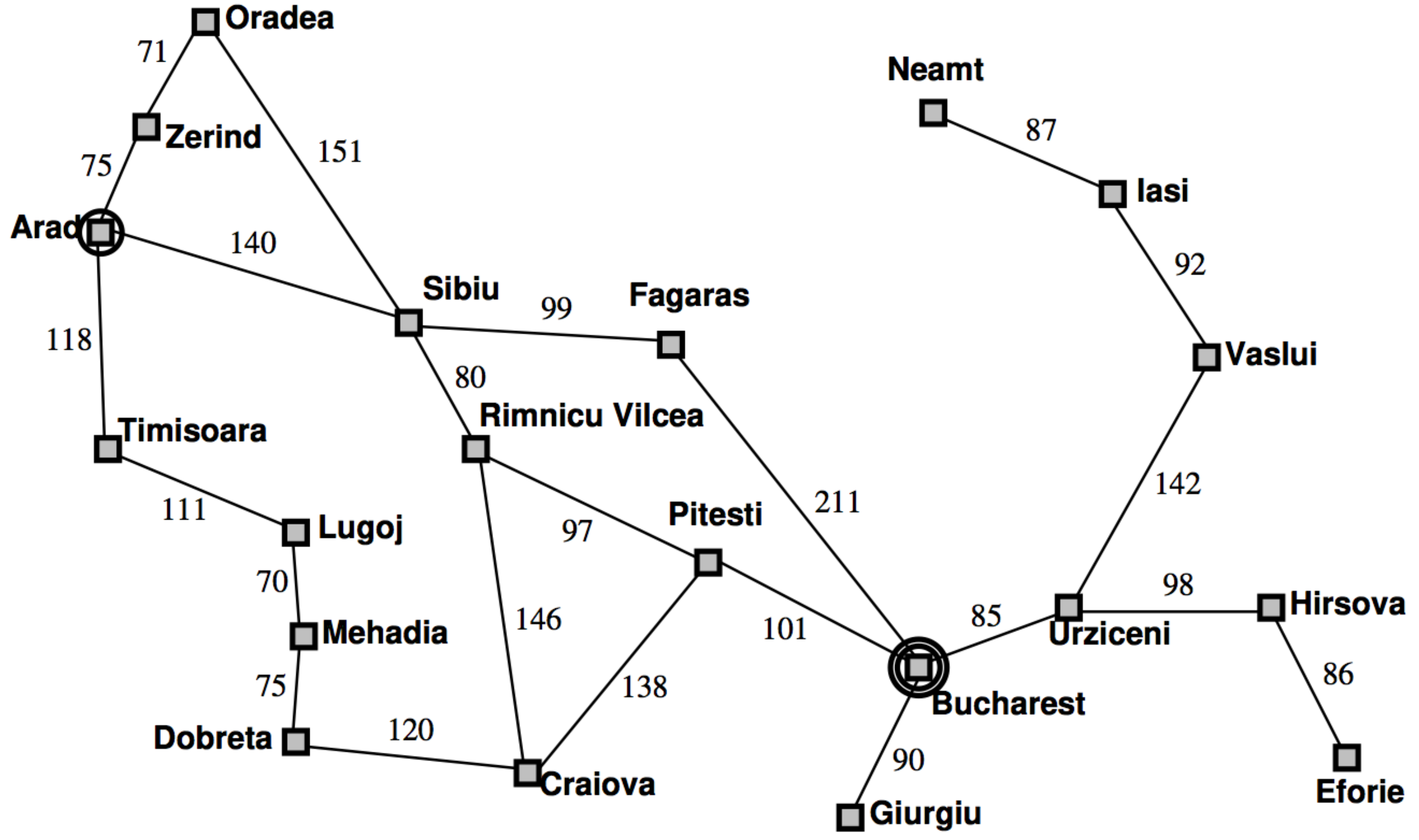
Search heuristics

A heuristic is:

- A function that estimates how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for path finding



Example



Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Stright-line distances to Bucharest

Greedy Search

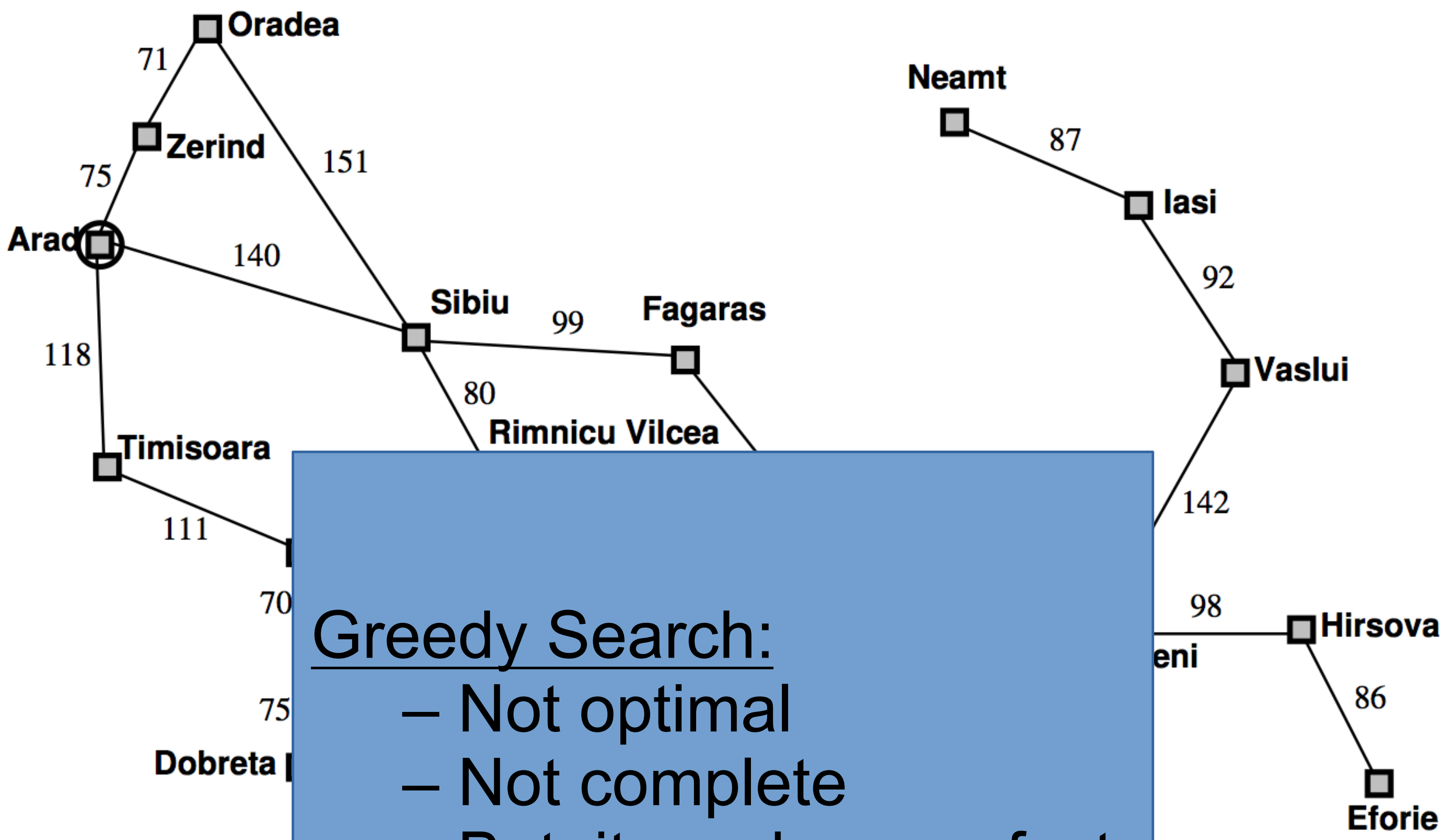
This is like a guess about how far the state is from the goal

Each time you expand a state, calculate the heuristic for each of the states that you add to the fringe.

– heuristic: $h(s)$ ⇐ i.e. distance to Bucharest

– on each step, choose to expand the state with the lowest heuristic value.

Example: Greedy Search



Greedy Search:

- Not optimal
- Not complete
- But, it can be very fast

Notice that this is not the optimal path!

Greedy vs UCS

<u>Greedy Search:</u>

- Not optimal
- Not complete
- But, it can be very fast

<u>UCS:</u>

- Optimal
- Complete
- Usually very slow

A*

s : a state

$g(s)$: minimum cost from start to s

$h(s)$: heuristic at s (*i.e.* an estimate of remaining cost-to-go)

UCS: expand states in order of $g(s)$

Greedy: expand states in order of $h(s)$

A*: expand states in order of $f(s) = g(s) + h(s)$

When is A* optimal?

It depends on whether we are using the tree search or the graph search version of the algorithm.

Optimal if *h* is consistent

– *h*(*s*) is an underestimate of the cost of each arc.

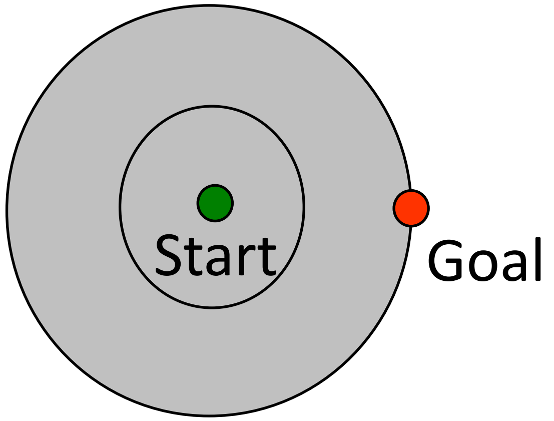
Optimal if *h* is admissible

– *h*(*s*) is an underestimate of the true cost-to-go.

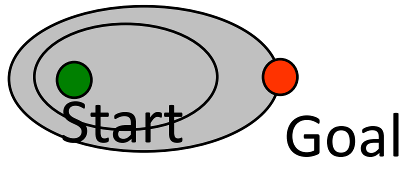
What is "cost-to-go"?
– minimum cost required to reach a goal state

A* vs UCS

Uniform-cost expands equally in all “directions”



A* expands mainly toward the goal, but does hedge its bets to ensure optimality



Problem set question

AIMA 3.14. Which of the following are true and which are false? Explain your answers.

(a) Depth-first search always expands at least as many nodes as A* search with an admissible heuristic.

(b) h(n) = 0 is an admissible heuristic for the 8-puzzle.

(c) A* is of no use in robotics because percepts, states, and actions are continuous.

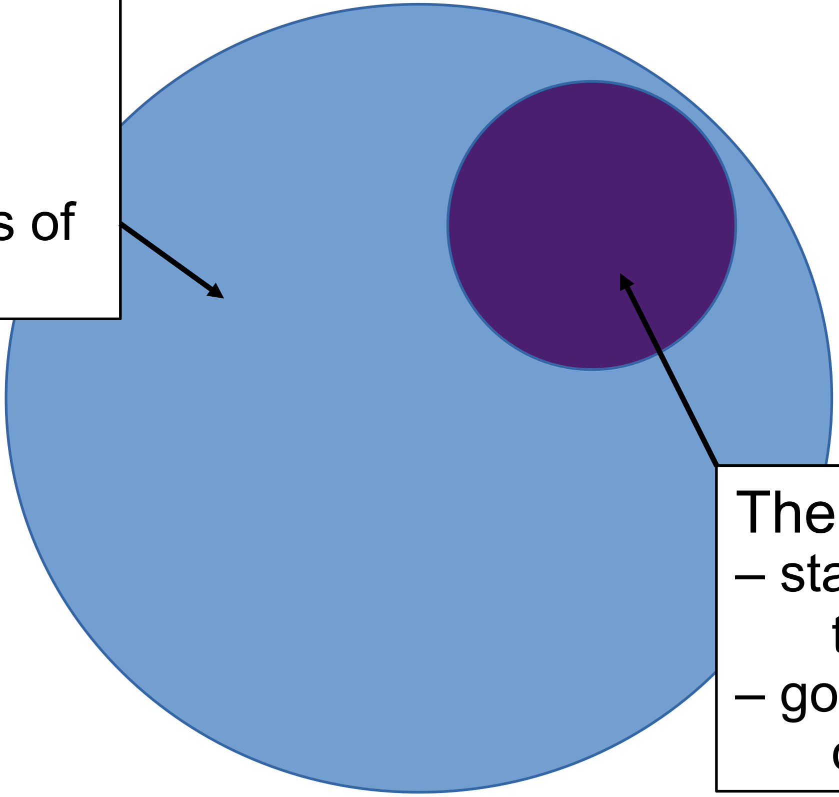
(d) Breadth-first search is complete even if zero step costs are allowed.

(e) Assume that a rook can move on a chessboard any number of squares in a straight line, vertically or horizontally, but cannot jump over other pieces. Manhattan distance is an admissible heuristic for the problem of moving the rook from square A to square B in the smallest number of moves.

Constraint Satisfaction

What is a CSP?

CSPs $\subseteq$ All search problems

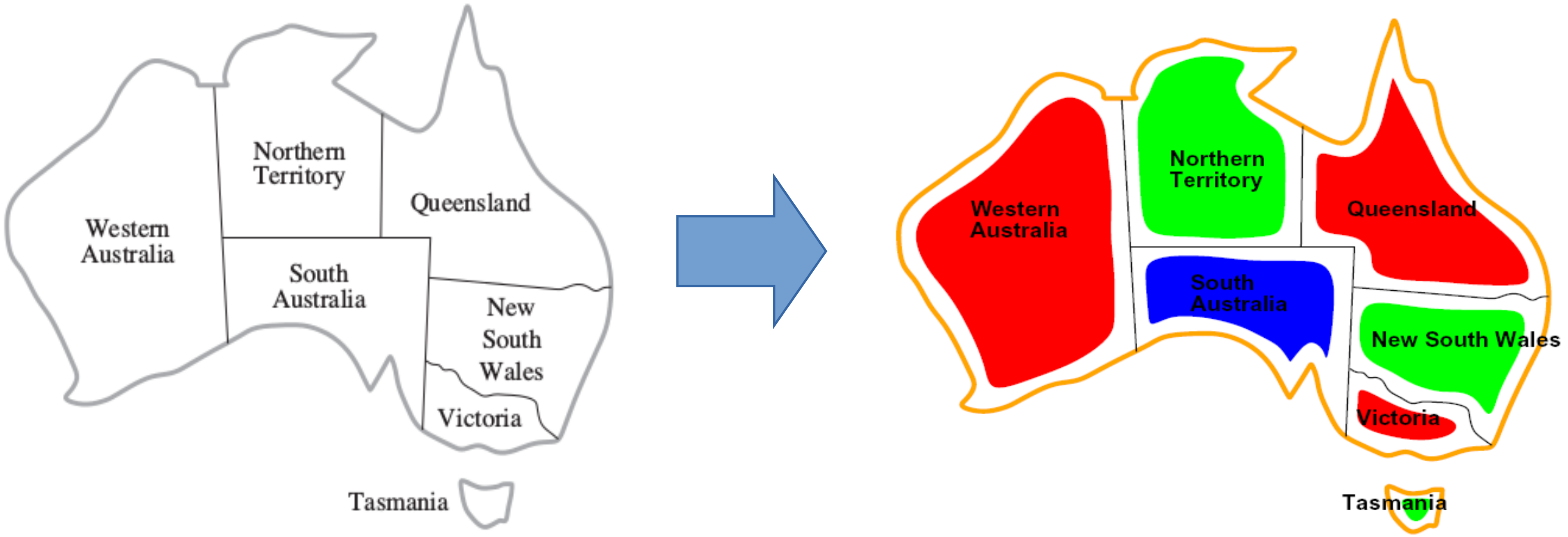


The space of all search problems
– states and actions are atomic
– goals are arbitrary sets of states

The space of all CSPs
– states are defined in terms of variables
– goals are defined in terms of constraints

<ins>A CSP is defined by:</ins>
1. a set of variables and their associated domains.
2. a set of constraints that must be satisfied.

CSP example: map coloring



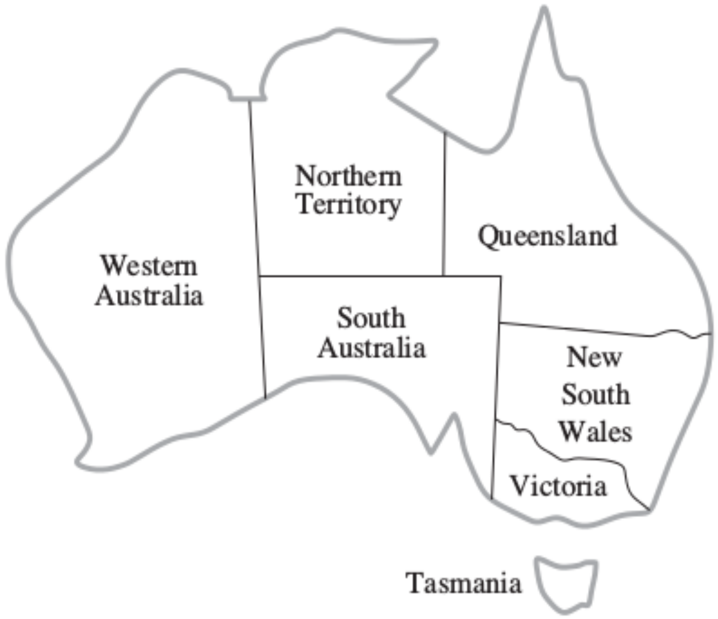
Problem: assign each territory a color such that no two adjacent territories have the same color

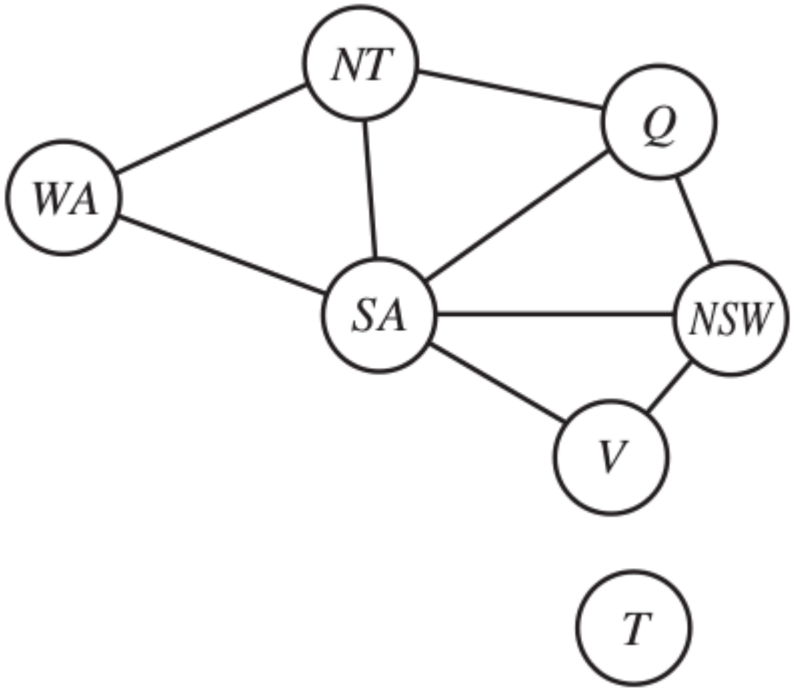
Variables: $X = \{WA, NT, Q, NSW, V, SA, T\}$

Domain of variables: $D = \{r, g, b\}$

Constraints: $C = \{SA \neq WA, SA \neq NT, SA \neq Q, \ldots\}$

The constraint graph





Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints

General-purpose CSP algorithms use the graph structure to speed up search
    E.g., Tasmania is an independent subproblem!

A harder CSP to represent: Cryptarithmetic

- Variables:

$F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

- Domains:

$\{0,1,2,3,4,5,6,7,8,9\}$

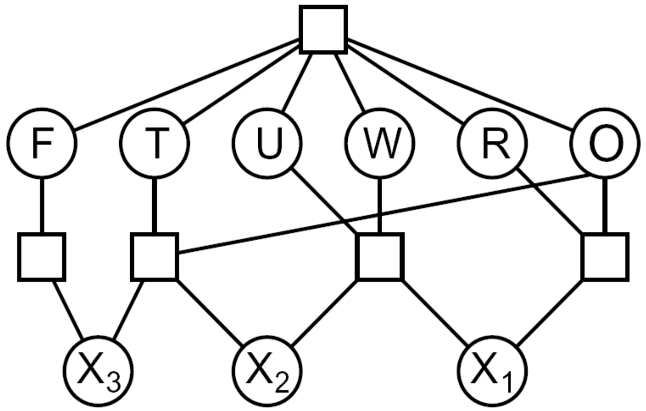
- Constraints:

$\text{alldiff}(F,T,U,W,R,O)$

$O + O = R + 10 \cdot X_1$

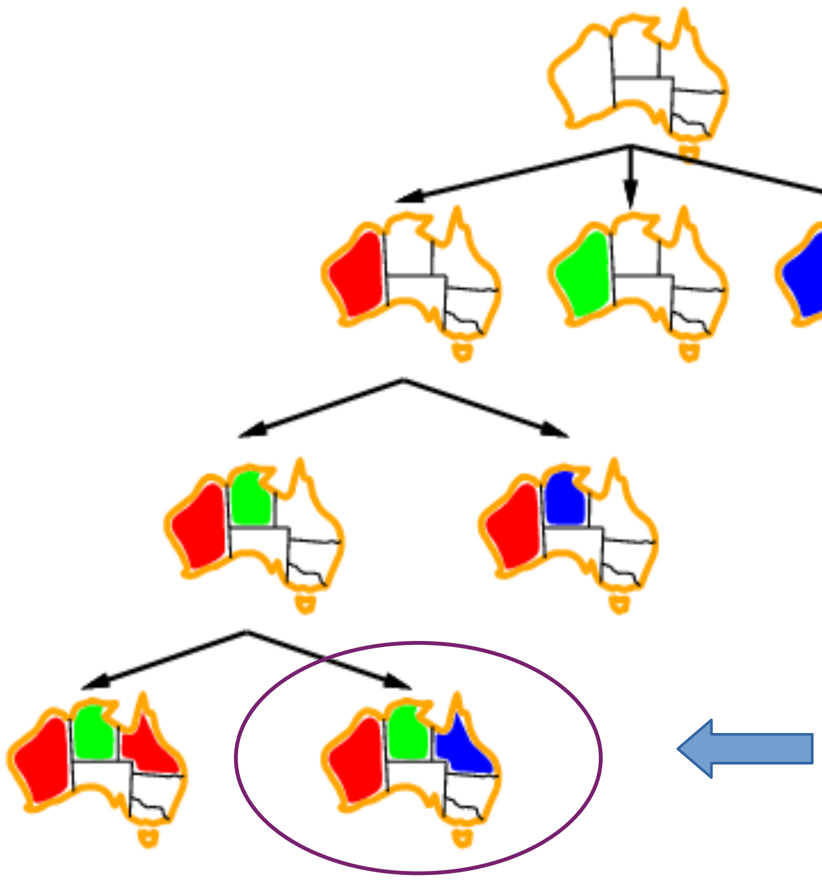
$\ldots$

```
    T W O
+   T W O
---------
  F O U R
```



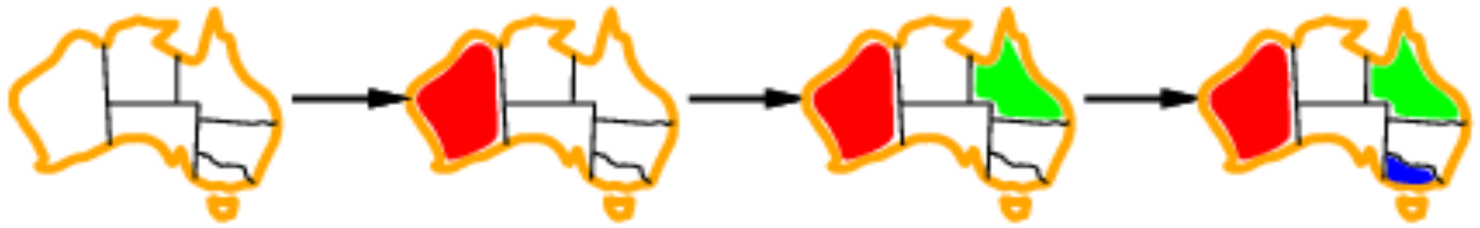
Backtracking search

When a node is expanded, check that each successor state is consistent before adding it to the queue.



Does this state have any valid successors?

Forward checking

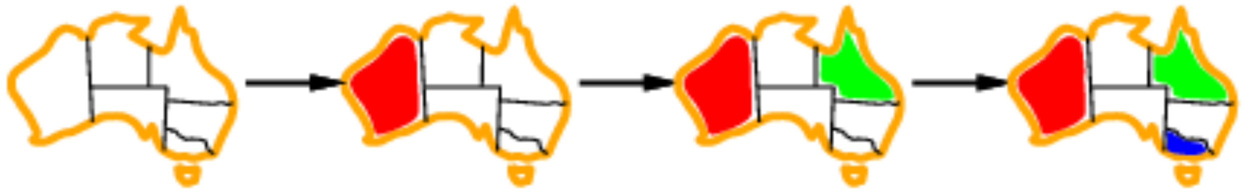


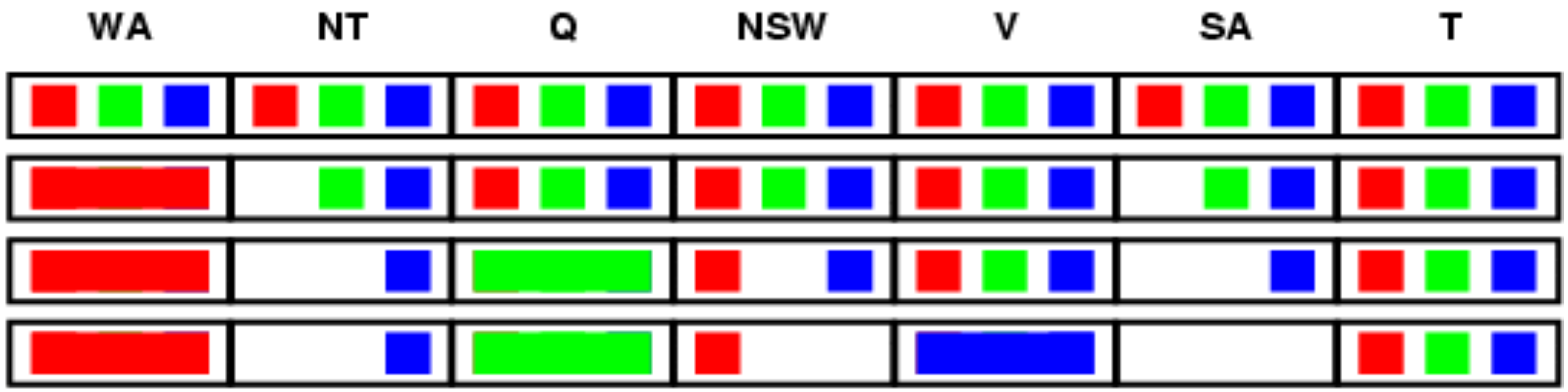
WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥	🟦		🟥 🟩 🟦

Track domain for each unassigned variable
- – initialize w/ domains from problem statement
- – each time you expand a node, update domains of all unassigned variables

Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:





WA	NT	Q	NSW	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥	🟦		🟥 🟩 🟦

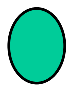
NT and SA cannot both be blue!

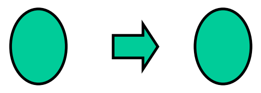
Constraint propagation repeatedly enforces constraints locally

K-consistency

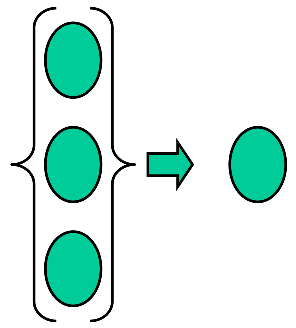
Increasing degrees of consistency

- 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints



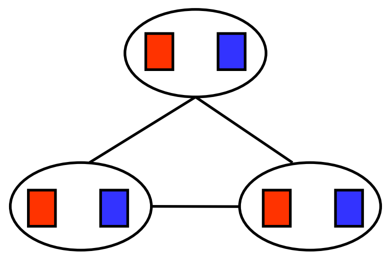


- 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other



- K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the kth node.

Higher k more expensive to compute

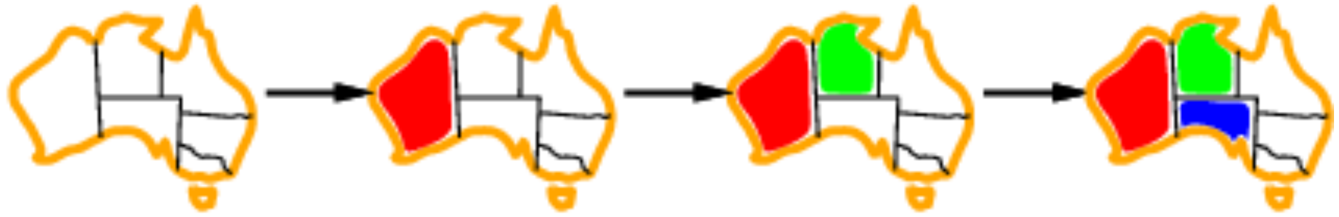


(You need to know the k=2 case: arc consistency)

Heuristics for improving CSP performance

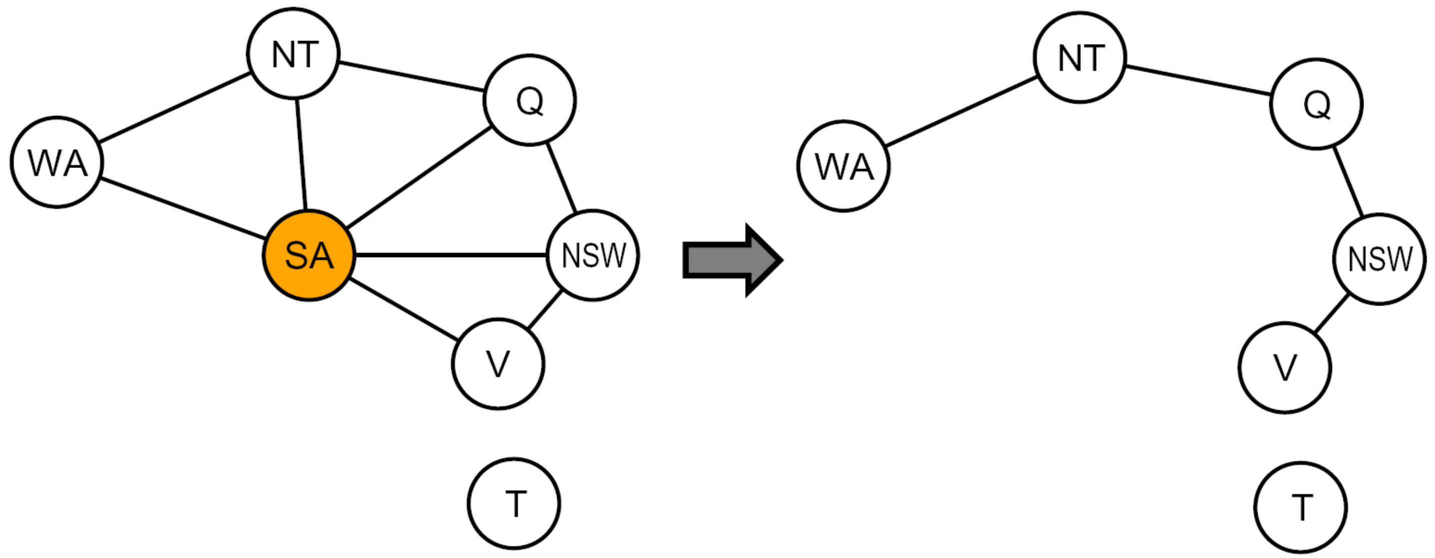
Minimum remaining values (MRV) heuristic:

– expand variables w/ minimum size domain first



Other heuristics such as the degree heuristic and least constraining value (LCV) heuristic.

Nearly tree-structured CSPs



Conditioning: instantiate a variable, prune its neighbors' domains

Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

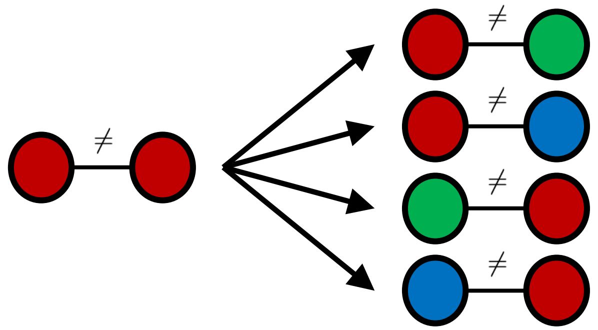
Cutset size c gives runtime $O((d^c)(n\text{-}c)d^2)$, very fast for small c

Aside: Local search more generally

Tree search keeps unexplored alternatives on the fringe (ensures completeness)

Local search: improve a single option until you can't make it better (no fringe!)

New successor function: local changes



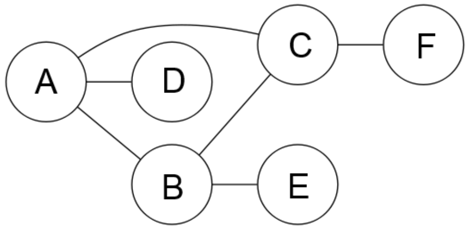
Generally much faster and more memory efficient (but incomplete and suboptimal)

Many local search algorithms (that we won't cover): hill climbing, simulated annealing, genetic algorithms, etc.

Berkeley question

The graph below is a constraint graph for a CSP that has only binary constraints. Initially, no variables have been assigned.

For each of the following scenarios, mark all variables for which the specified filtering might result in their domain being changed.



(i) [1 pt] A value is assigned to A. Which domains might be changed as a result of running forward checking for A?

○ A ○ B ○ C ○ D ○ E ○ F

(ii) [1 pt] A value is assigned to A, and then forward checking is run for A. Then a value is assigned to B. Which domains might be changed as a result of running forward checking for B?

○ A ○ B ○ C ○ D ○ E ○ F

(iii) [1 pt] A value is assigned to A. Which domains might be changed as a result of enforcing arc consistency after this assignment?

○ A ○ B ○ C ○ D ○ E ○ F

(iv) [1 pt] A value is assigned to A, and then arc consistency is enforced. Then a value is assigned to B. Which domains might be changed as a result of enforcing arc consistency after the assignment to B?

○ A ○ B ○ C ○ D ○ E ○ F

Adversarial search

Different types of games

Deterministic / stochastic

Two player / multi player?

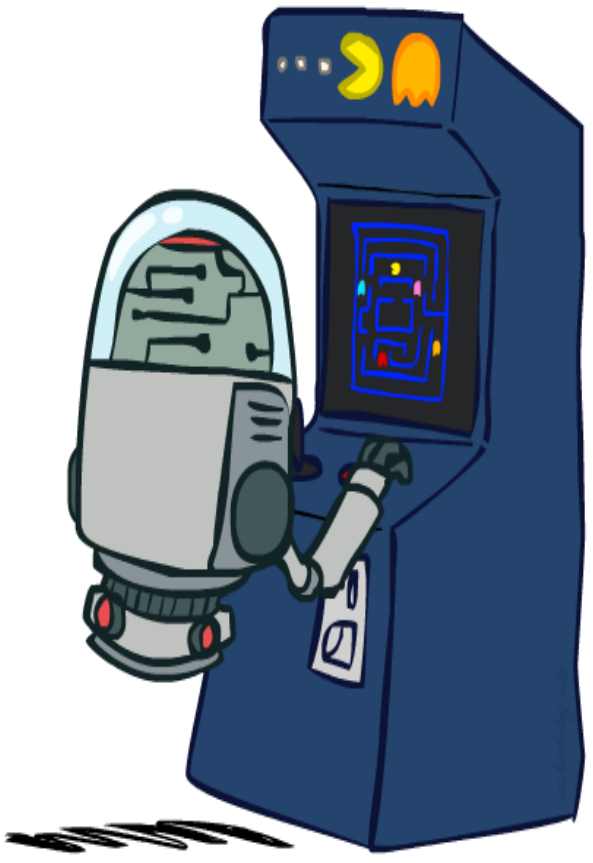
Zero-sum / non zero-sum

Perfect information / imperfect information

Deterministic games

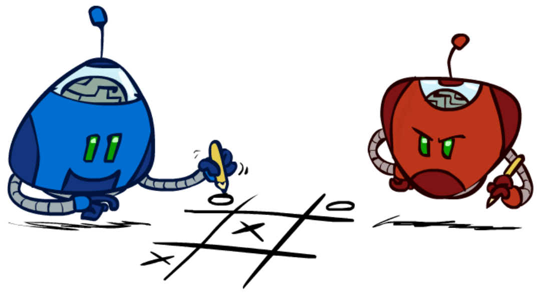
Many possible formalizations, one is:

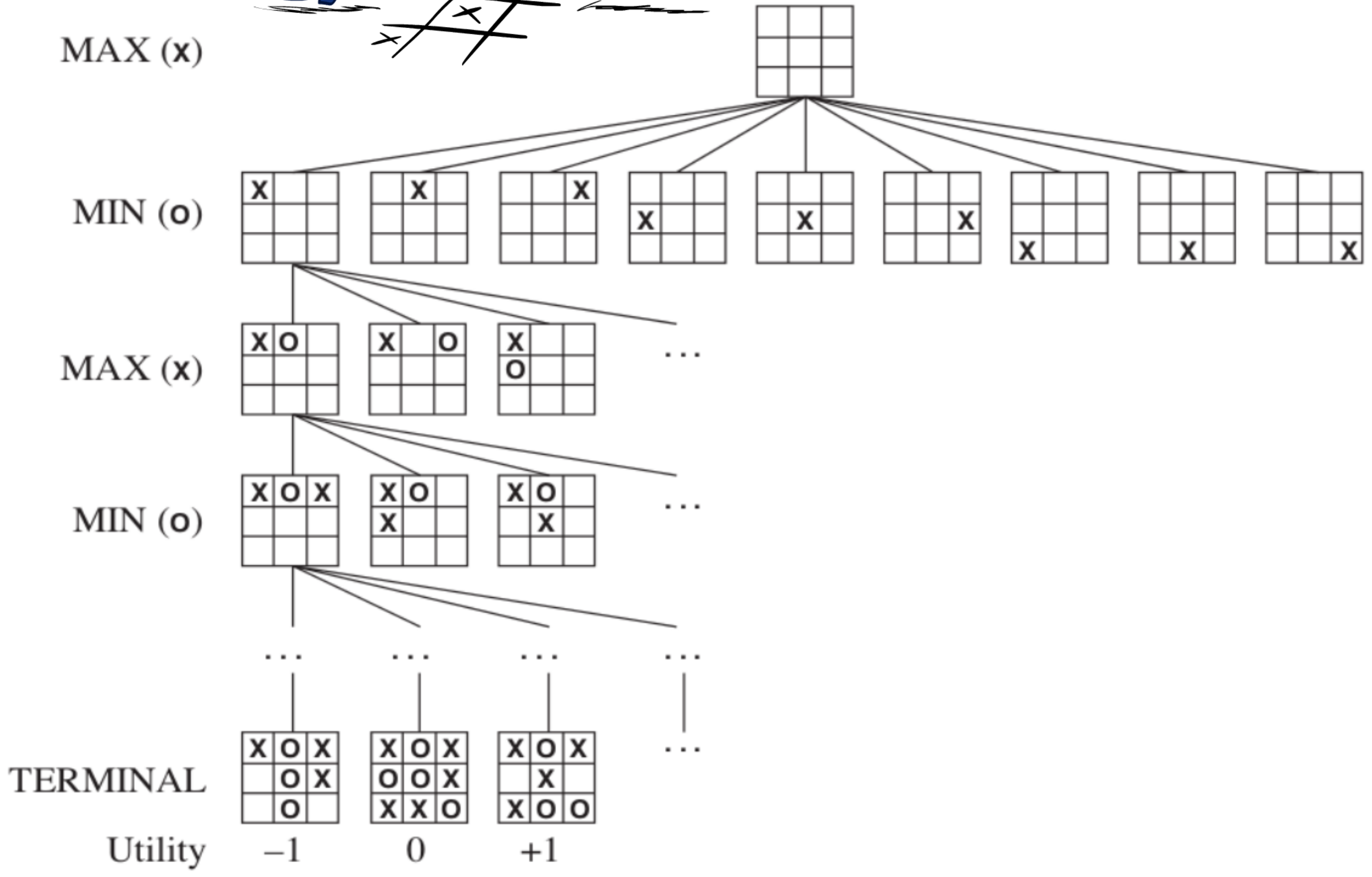
- States: S (start at s_0)
- Players: $P=\{1...N\}$ (usually take turns)
- Actions: A (may depend on player / state)
- Transition Function: $S \times A \rightarrow S$
- Terminal Test: $S \rightarrow \{t,f\}$
- Terminal Utilities: $S \times P \rightarrow R$

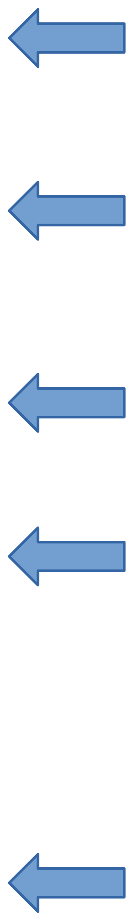


Solution for a player is a policy: $S \rightarrow A$

This is a game tree for tic-tac-toe







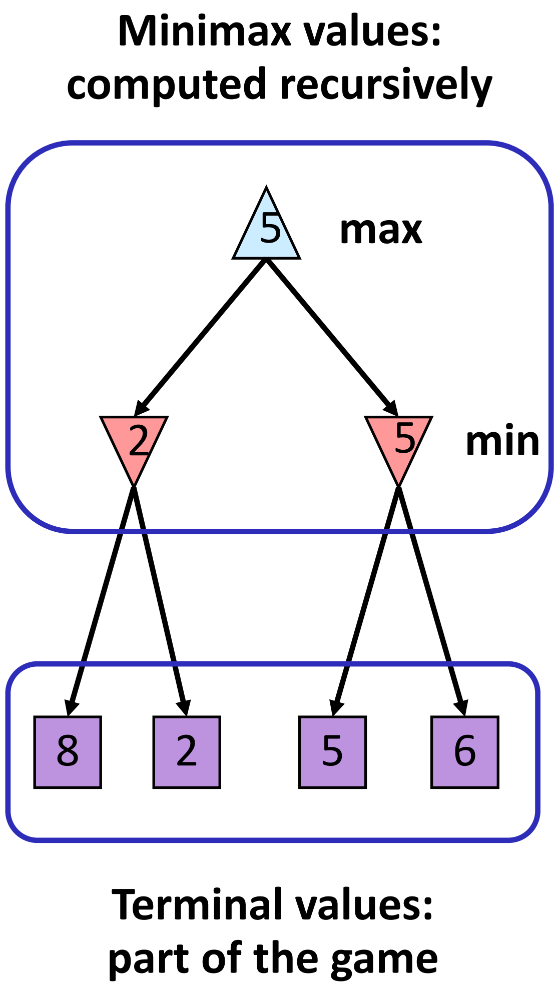
Minimax

Deterministic, zero-sum games:

- Tic-tac-toe, chess, checkers
- One player maximizes result
- The other minimizes result

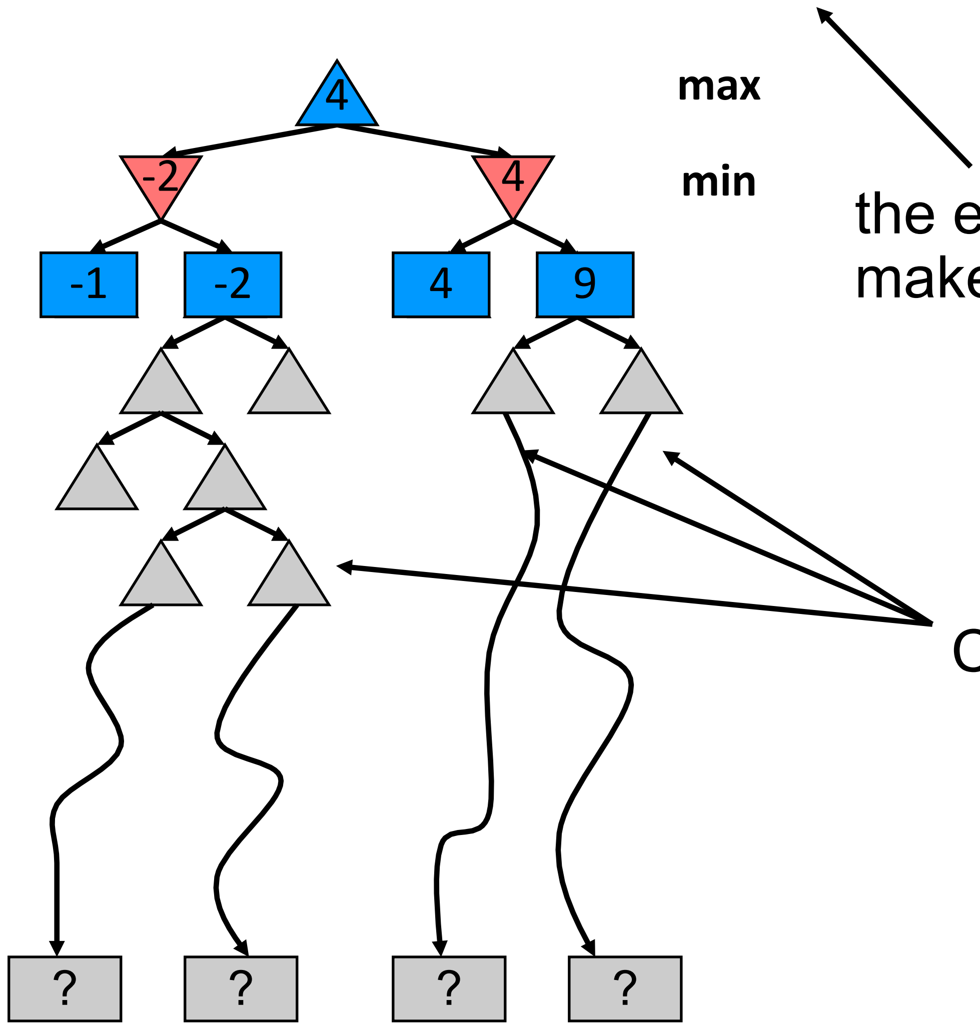
Minimax search:

- A state-space search tree
- Players alternate turns
- Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary


Minimax values: computed recursively
5
max
2
5
min
8
2
5
6
Terminal values: part of the game

Evaluation functions

Key idea: cut off search at a certain depth and give the corresponding nodes an estimated value.



the evaluation function makes this estimate.

Cut it off here

Alpha/Beta pruning: algorithm

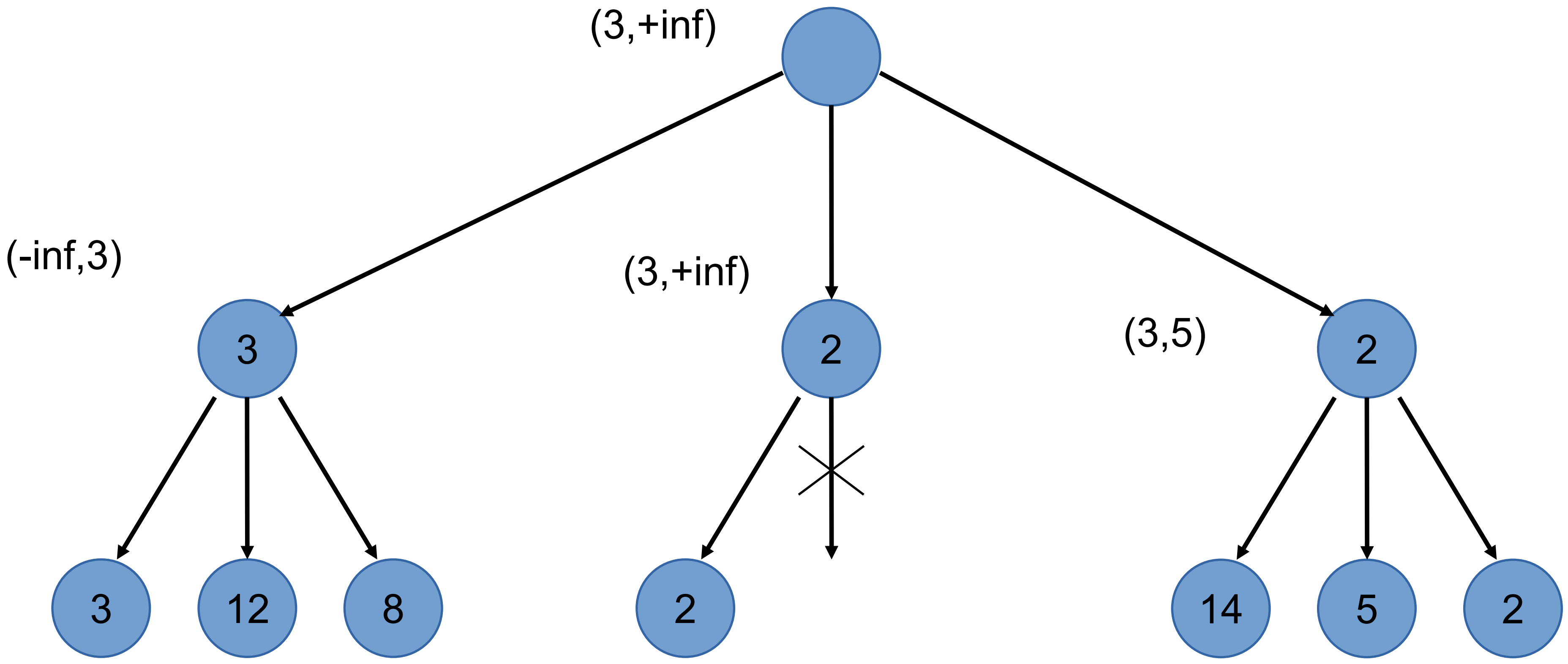
α: MAX's best option on path to root

β: MIN's best option on path to root

```
def max-value(state, $\alpha$, $\beta$):
    initialize v = $-\infty$
    for each successor of state:
        v = max(v, value(successor, $\alpha$, $\beta$))
        if $v \geq \beta$ return v
        $\alpha$ = max($\alpha$, v)
    return v
```

```
def min-value(state , $\alpha$, $\beta$):
    initialize v = $+\infty$
    for each successor of state:
        v = min(v, value(successor, $\alpha$, $\beta$))
        if $v \leq \alpha$ return v
        $\beta$ = min($\beta$, v)
    return v
```

Alpha/Beta pruning



Expectimax search

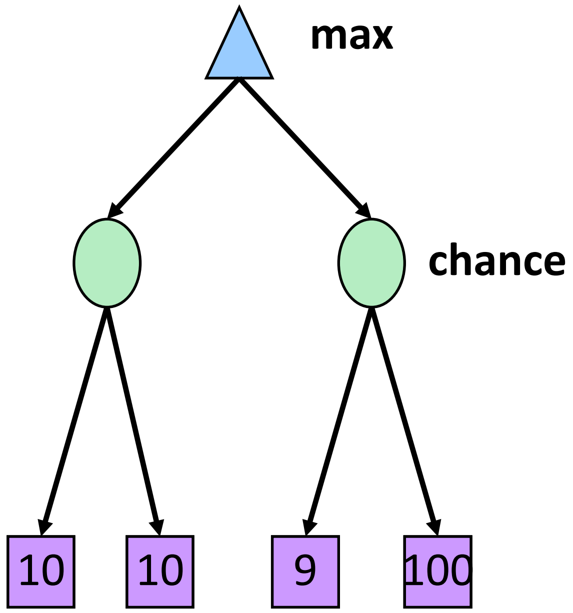
Why wouldn't we know the result of an action?

- Explicit randomness: rolling dice
- Unpredictable opponents: the ghosts respond randomly
- Actions can fail: when moving a robot, wheels may slip

Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes

Expectimax search: compute the average score under optimal play

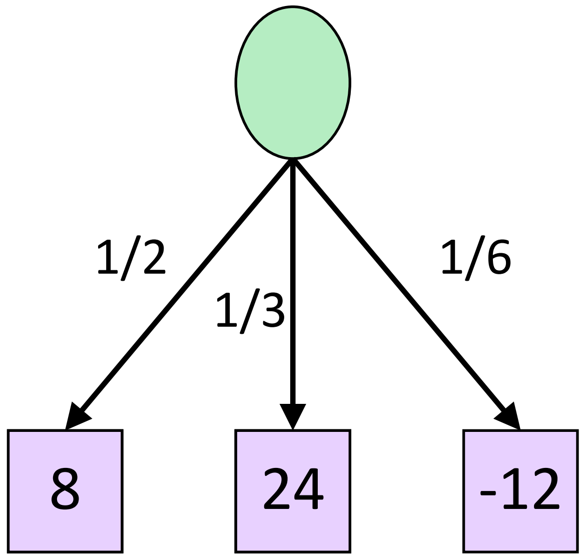
- Max nodes as in minimax search
- Chance nodes are like min nodes but the outcome is uncertain
- Calculate their expected utilities
- I.e. take weighted average (expectation) of children



Later, we'll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes

Expectimax pseudocode

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```



$v = (1/2)\ (8) + (1/3)\ (24) + (1/6)\ (-12) = 10$

Problem set question

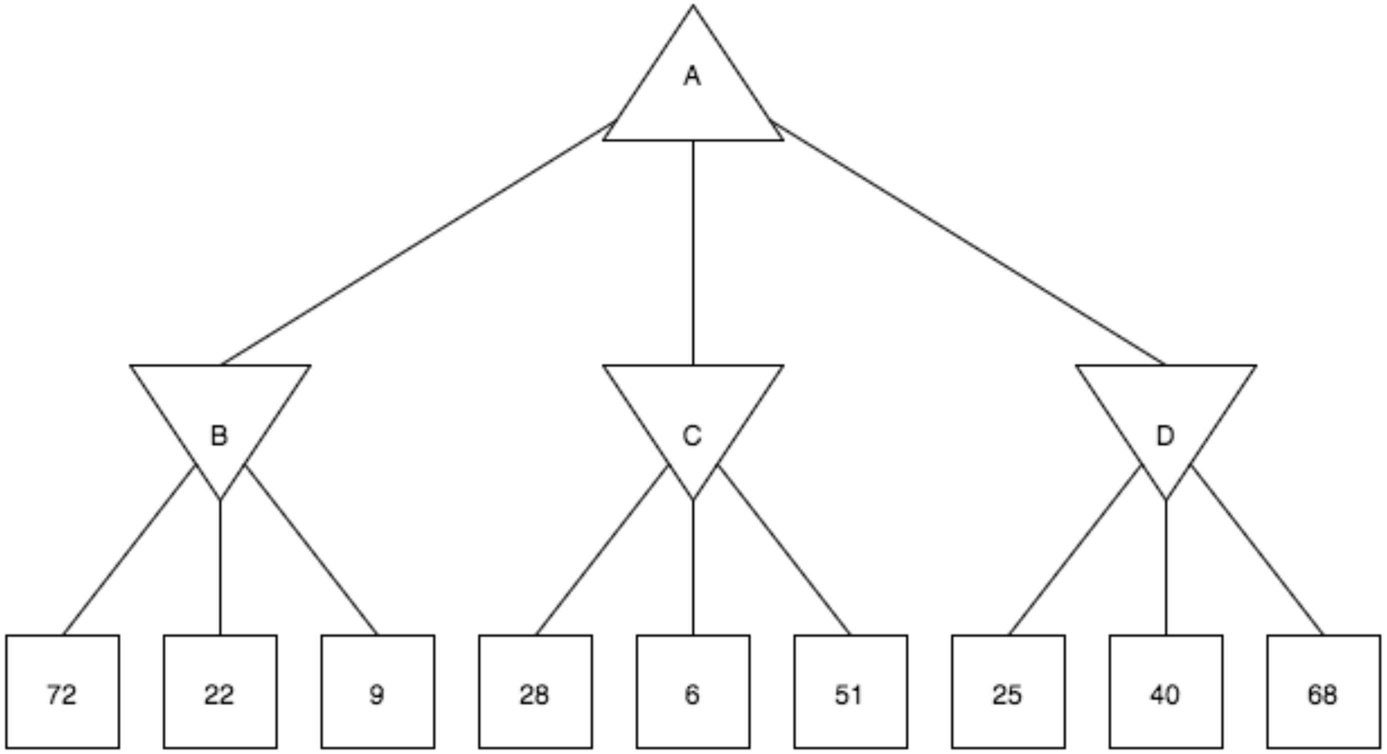


Figure 1: Minimax

(a) For the Minimax tree above, what is the value of A,B,C,D?

Problem set question

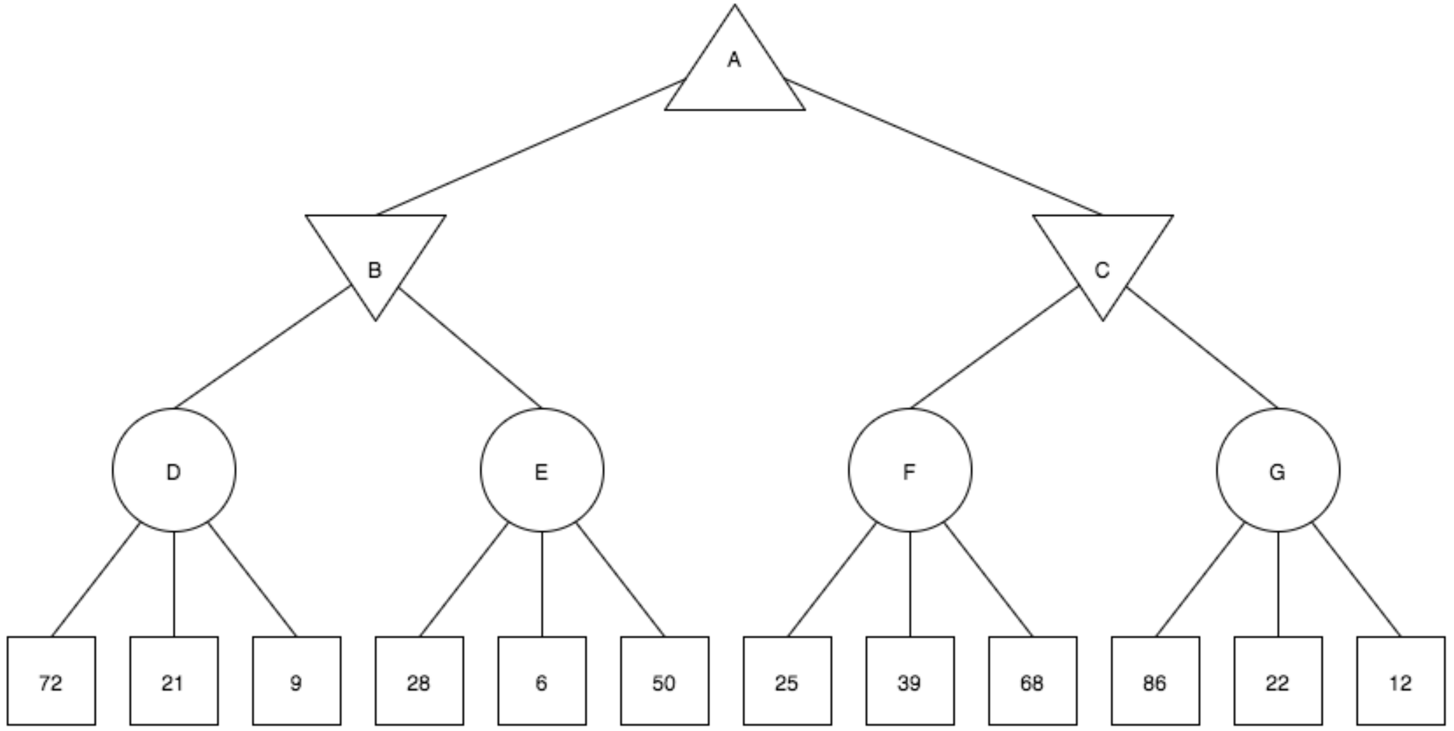


Figure 2: Expectimax

(a) For the Expectimax tree above, what is the value of A,B,C,D,E,F,G? (Assume uniform random probabilities)

Problem set question

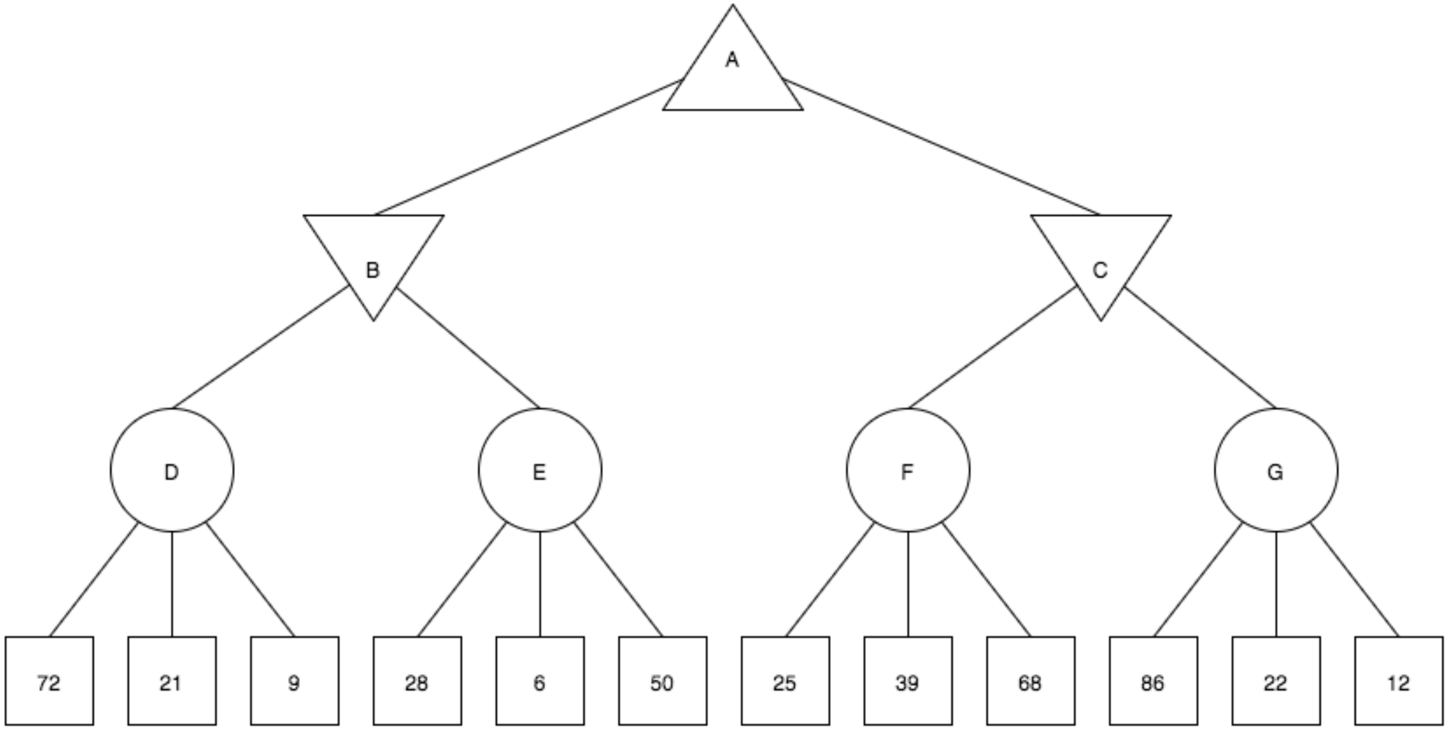


Figure 2: Expectimax

(a) For the Expectimax tree above, what is the value of A,B,C,D,E,F,G? (Assume uniform random probabilities)

A:max(28, 40) = 40 B:min(34, 28) = 28 C:min(44, 40) = 40
D:(72 + 21 + 9)/3 = 34 E:(28 + 6 + 50)/3 = 28 F:(25 + 39 + 68)/3 = 44 G:(86 + 22 + 12)/3 = 40

Probability

(Discrete) random variables

What is a random variable?

Discrete random variable, X, can take on many (possibly infinite) values, called the *state space* or *domain* $A=\{1,2,3,4,5,6\}$ (e.g., a die)

$$a \in \underbrace{\{1, 2, 3, 4, 5, 6\}} = A$$

a is a *random variable*

this is the *domain* of *a*

Another example:

Suppose *b* denotes whether it is raining or clear outside:

$$b \in \{rain, clear\} = B$$

Probability distribution

A probability distribution associates each with a probability of occurrence, represented by a *probability mass function (pmf)*.

A probability table is one way to encode the distribution:

$$a \in \{1, 2, 3, 4, 5, 6\} = A \qquad b \in \{rain, clear\} = B$$

a	P(a)
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

b	P(b)
rain	1/4
clear	3/4

All probability distributions must satisfy the following:

1. $\forall a \in A, a \geq 0$
2. $\sum_{a \in A} a = 1$

Joint probability distributions

Given random variables: $X_1, X_2, \ldots, X_n$

The *joint distribution* is a probability assignment to all combinations: $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$

or: $P(x_1, x_2, \ldots, x_n)$

Sometimes written as: $P(X_1 = x_1 \wedge X_2 = x_2 \wedge \ldots \wedge X_n = x_n)$

As with single-variate distributions, joint distributions must satisfy:

1. $P(x_1, x_2, \ldots, x_n) \geq 0$

2. $\displaystyle\sum_{x_1, \ldots, x_n} P(x_1, x_2, \ldots, x_n) = 1$

Prior or unconditional probabilities of propositions
e.g., P (*Cavity* = *true*) = 0.1 and P (*Weather* = *sunny*) = 0.72
correspond to belief prior to arrival of any (new) evidence

Marginalization

Given P(T,W), calculate P(T) or P(W)...

T	W	P(T,W)
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$$P(T) = \sum_{w \in W} P(T, w)$$

T	P(T)
hot	0.5
cold	0.5

$$P(W) = \sum_{t \in T} P(t, W)$$

W	P(W)
sun	0.5
rain	0.4

Conditional Probabilities

Conditional probability: $P(A \mid B) = \frac{P(A,B)}{P(B)}$ (if P(B)>0)

Example: Medical diagnosis

Product rule: P(A,B) = P(A $\wedge$ B) = P(A|B)P(B)

Marginalization with conditional probabilities:

$$P(A) = \sum_{b \in B} P(A \mid B = b) P(B = b)$$

This formula/rule is called the *law of of total probability*

Chain rule is derived by successive application of product rule:
$P(X_1,\ldots,X_n) = P(X_1,\ldots,X_{n-1})\, P(X_n|X_1,\ldots,X_{n-1})$
$= P(X_1,\ldots,X_{n-2})\, P(X_{n-1}|X_1,\ldots,X_{n-2})\, P(X_n|X_1,\ldots,X_{n-1}) = \ldots$
$= \Pi^n_{i=1} P(X_i|X_1,\ldots,X_{i-1})$

Normalization

T	W	P(T,W)
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$\longrightarrow$

W	P(W, t=hot)
sun	0.4
rain	0.1

$\longrightarrow$

W	$P(W \mid t = hot)$
sun	0.8
rain	0.2

Select corresponding elts from the joint distribution

Scale the numbers so that they sum to 1.

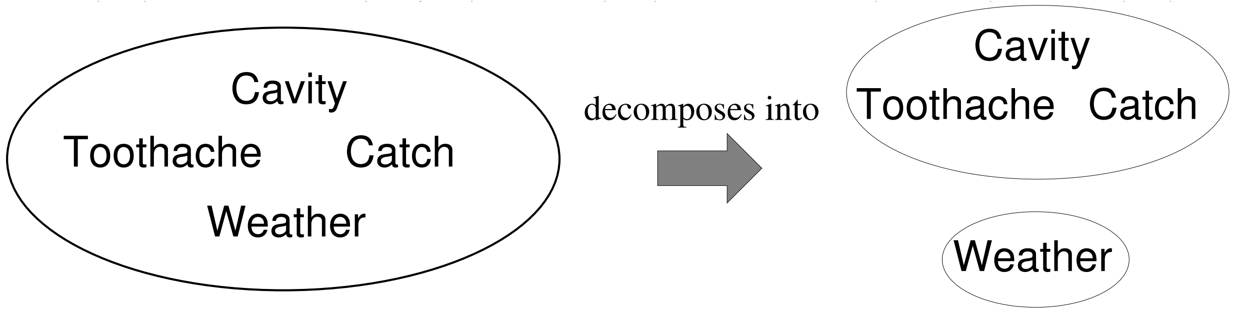
$$P(sun|cold) = \frac{P(sun, cold)}{P(cold)} = \frac{P(sun, cold)}{\boxed{P(sun, cold) + P(rain, cold)}}$$

The only purpose of this denominator is to make the distribution sum to one.
– we achieve the same thing by scaling.

Independence

A and *B* are *independent* iff
$P(A|B) = P(A)$ or $P(B|A) = P(B)$ or $P(A,B) = P(A)P(B)$

$P(Toothache, Catch, Cavity, Weather)$
$= P(Toothache, Catch, Cavity)P(Weather)$



32 entries reduced to 12; for *n* independent biased coins, $2^n \rightarrow n$

Absolute independence powerful but rare

Dentistry is a large field with hundreds of variables, none of which are independent. What to do?

Conditional independence

P(Toothache, Cavity, Catch) has 23 − 1 = 7 independent entries

If I have a cavity, the probability that the probe catches in it doesn't depend on whether I have a toothache:

(1)P(*catch*|*toothache, cavity*) = *P*(*catch*|*cavity*)

The same independence holds if I haven't got a cavity:

(2) P(*catch*|*toothache*, ¬*cavity*) = P(*catch*|¬*cavity*)

Catch is conditionally independent of *Toothache* given *Cavity*:

P(*Catch*|*Toothache*, *Cavity*) = P(*Catch*|*Cavity*)

Equivalent statements:

P(*Toothache*|*Catch, Cavity*)=P(*Toothache*|*Cavity*)

P(*Toothache*, *Catch*|*Cavity*)=P(*Toothache*|*Cavity*)P(*Catch*|*Cavity*)

Bayes' Rule

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

Thomas Bayes?

Using Bayes' Rule

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

$$\Downarrow$$

$$\underbrace{P(cause|effect)}_{\text{But harder to estimate this}} = \frac{\underbrace{\boxed{P(effect|cause)}}_{\text{It's often easier to estimate this}}P(cause)}{P(effect)}$$

Making decisions under uncertainty

Rational decision making requires reasoning about one’s *uncertainty* and *objectives*

Previous section focused on uncertainty

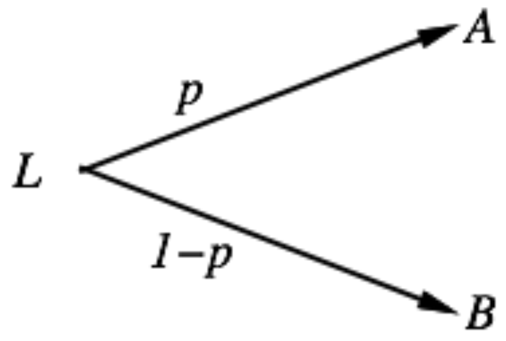
This section will discuss how to make rational decisions based
on a *probabilistic model* and *utility function*

Focus will be on single step decisions, next week we will consider sequential decision problems

Preferences

An agent chooses among prizes (A, B, etc.) and lotteries, i.e., situations with uncertain prizes

Lottery $L=[p,A;\ (1-p),B]$



Notation:

$A \succ B$ A preferred to B

$A \sim B$ indifference between A and B

$A \succsim B$ B not preferred to A

Preferences lead to utilities

Note: an agent can be entirely rational (consistent with MEU) without ever representing or manipulating utilities and probabilities

E.g., a lookup table for perfect tic-tac-toe

Although a utility function must exist, it is not unique

If $U'(S)=aU(S)+b$ and a and b are constants with $a>0$, then preferences of U' are the same as U

E.g., temperatures in Celcius, Fahrenheit, Kelvin

Maximizing expected utility (MEU)

Theorem (Ramsey, 1931; von Neumann and Morgenstern, 1944): Given preferences satisfying the constraints there exists a real-valued function U such that

$$U(A) \geq U(B) \Leftrightarrow A \succsim B$$

$$U(A) > U(B) \Leftrightarrow A \succ B$$

$$U(A) = U(B) \Leftrightarrow A \sim B$$

$$U([p_1, s_1; \ldots; p_n, s_n]) = \sum_i p_i U(s_i)$$

MEU principle: Choose the action that maximizes expected utility

Problem set question

AIMA 13.3 For each of the following statements, either prove it is true or give a counterexample.

(a) If $P(a|b,c) = P(b|a,c)$, then $P(a|c) = P(b|c)$

(b) If $P(a|b,c) = P(a)$, then $P(b|c) = P(b)$

(c) If $P(a|b) = P(a)$, then $P(a|b,c) = P(a|c)$

MDPs

What is an *MDP*?

Technically, an MDP is a 4-tuple

An MDP (Markov Decision Process) defines a stochastic control problem: $M = (S, A, T, R)$

State set: $s \in S$

Action Set: $a \in A$

Transition function: $T : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$

Probability of going from *s* to *s′* when executing action *a*

Reward function: $R : S \times A \rightarrow \mathbb{R}_{\geq 0}$

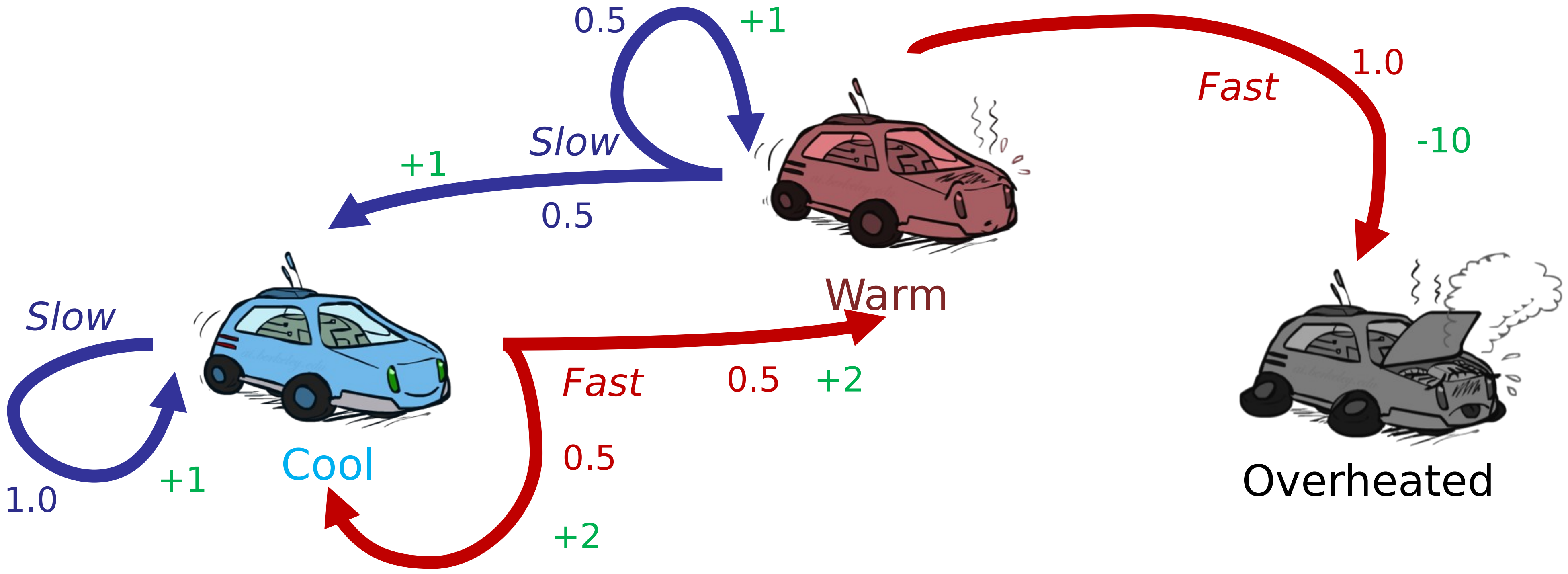
$$\sum_{s' \in S} T(s, a, s') = 1$$

<u>Objective</u>: calculate a strategy for acting so as to maximize the (discounted) sum of future rewards.

– we will calculate a *policy* that will tell us how to act

Example

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



Discounting rewards

In general:

$$\begin{aligned} U_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \end{aligned}$$

Utility

Models of optimal behavior

In the *finite-horizon* model, agent should optimize expected reward for the next *H* steps:

$$\mathbb{E}\left(\sum_{t=0}^{H} r_t\right)$$

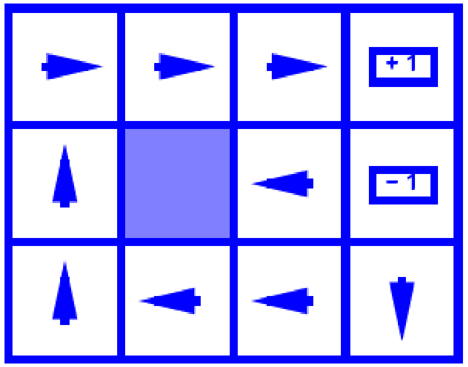
- Continuously executing *H*-step optimal actions is known as receding horizon control

In the *infinite-horizon* discounted model agent should optimize:

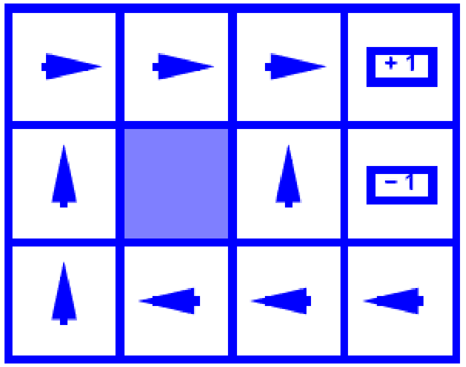
$$\mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

- Discount factor $0 \leq \gamma < 1$ can be thought of as an interest rate (reward now is worth more than reward in the future)

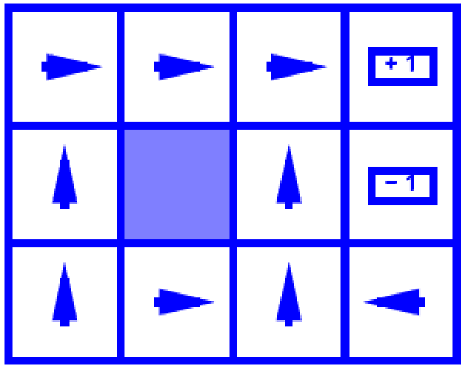
Examples of optimal policies



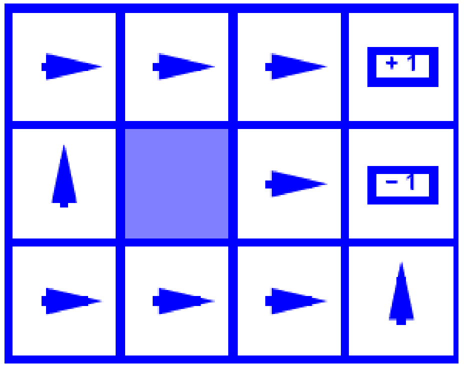
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Solving MDPs

- The value (utility) of a state s:

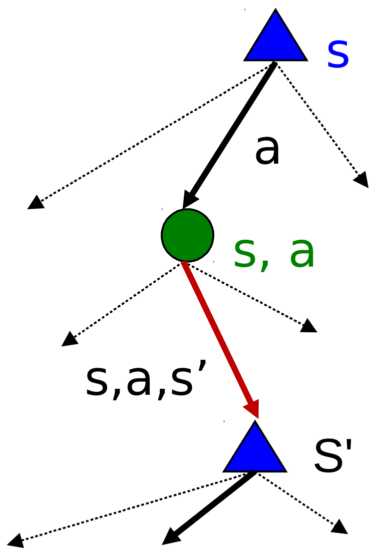
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s

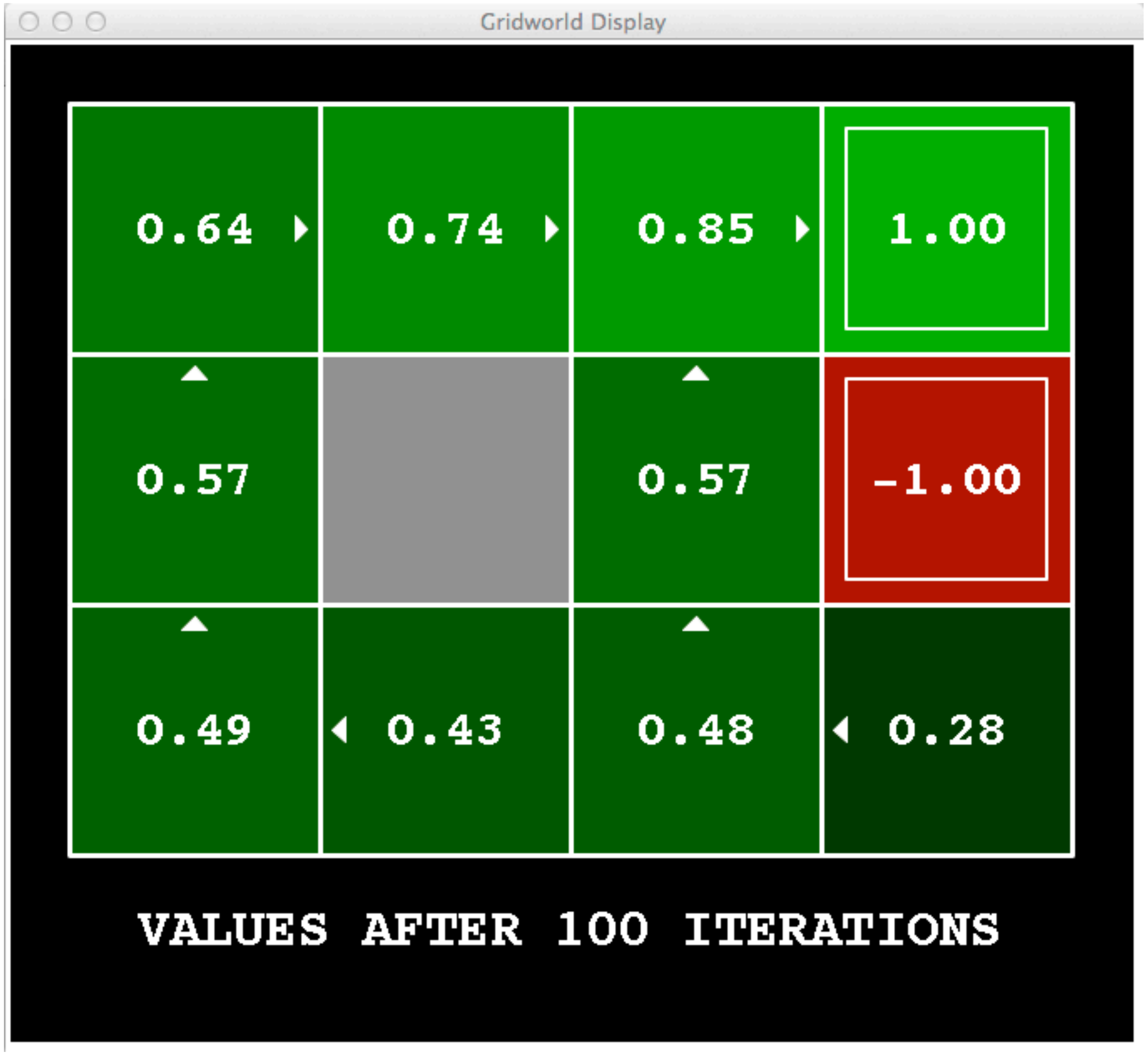


s is a *state*

(s, a) is a *q-state*

(s,a,s’) is a *transition*

Snapshot of Demo – Gridworld V Values

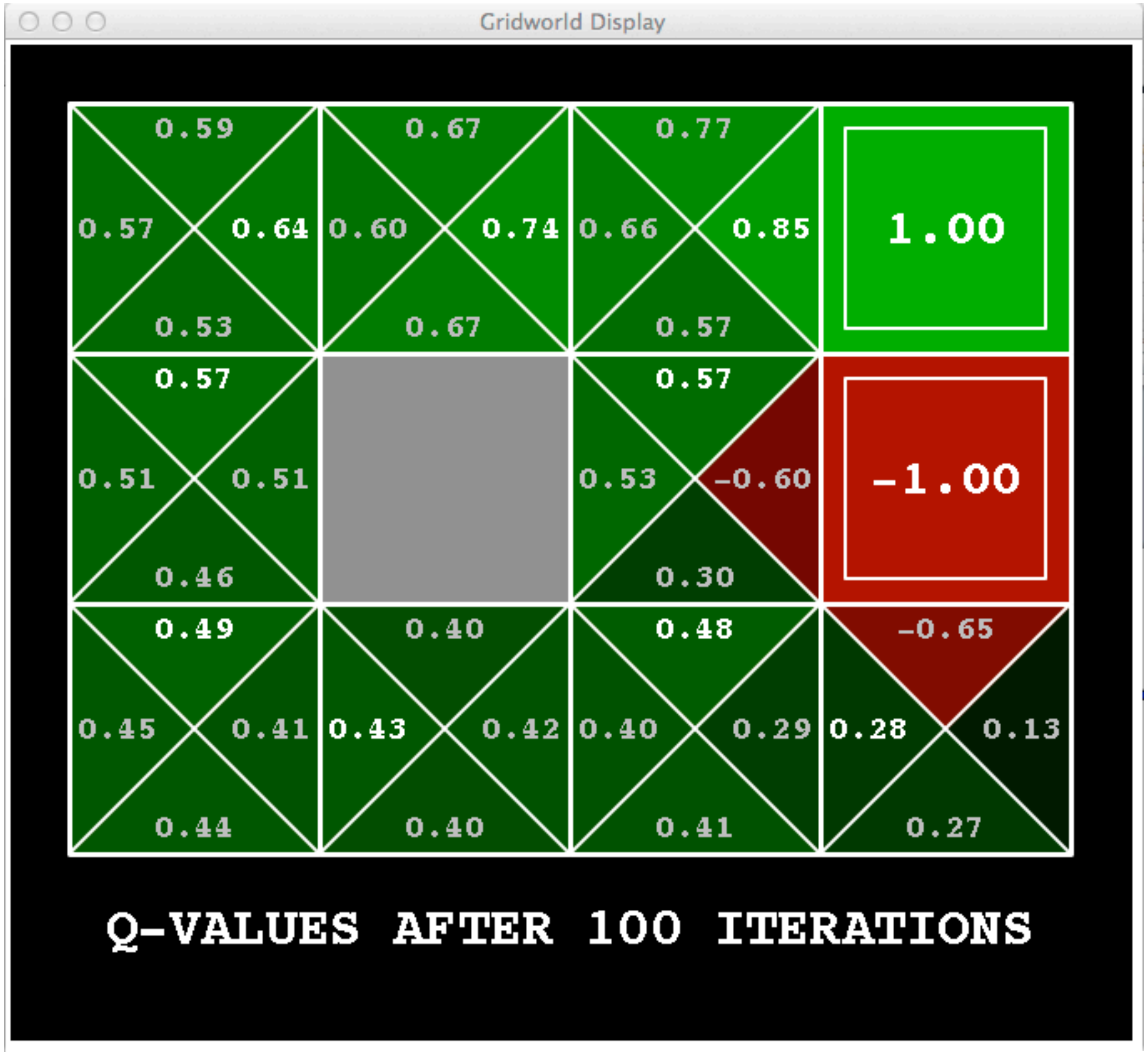


0.64 ▶	0.74 ▶	0.85 ▶	1.00
▲ 0.57		▲ 0.57	-1.00
▲ 0.49	◀ 0.43	▲ 0.48	◀ 0.28

VALUES AFTER 100 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

Snapshot of Demo – Gridworld V Values



Noise = 0.2
Discount = 0.9
Living reward = 0

Value iteration

Value of *s* at *k* timesteps to go: $V_k(s)$

Value iteration:

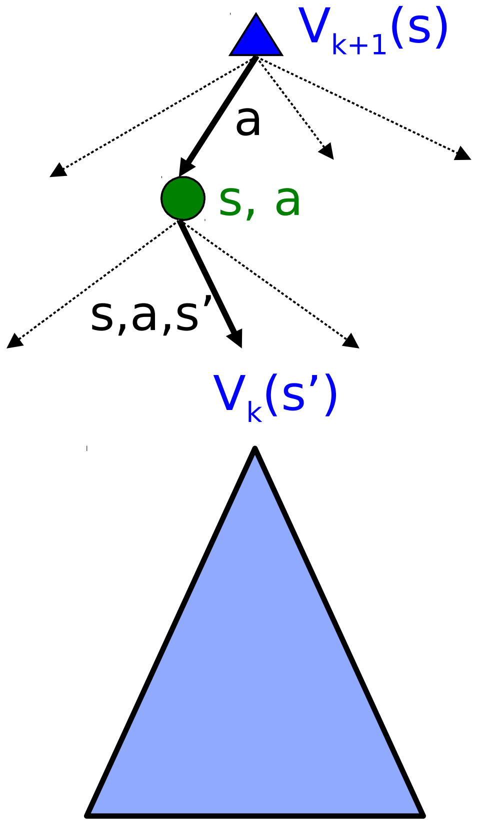
1. initialize $V_0(s) = 0$

2. $V_1(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_0(s') \right]$

3. $V_2(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_1(s') \right]$

4. ....

5. $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$



Bellman Equations and Value iteration

- Bellman equations characterize the optimal values:

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

- Value iteration computes them:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s')\right]$$

- Value iteration is just a fixed point solution method
  … though the V_k vectors are also interpretable as time-limited values

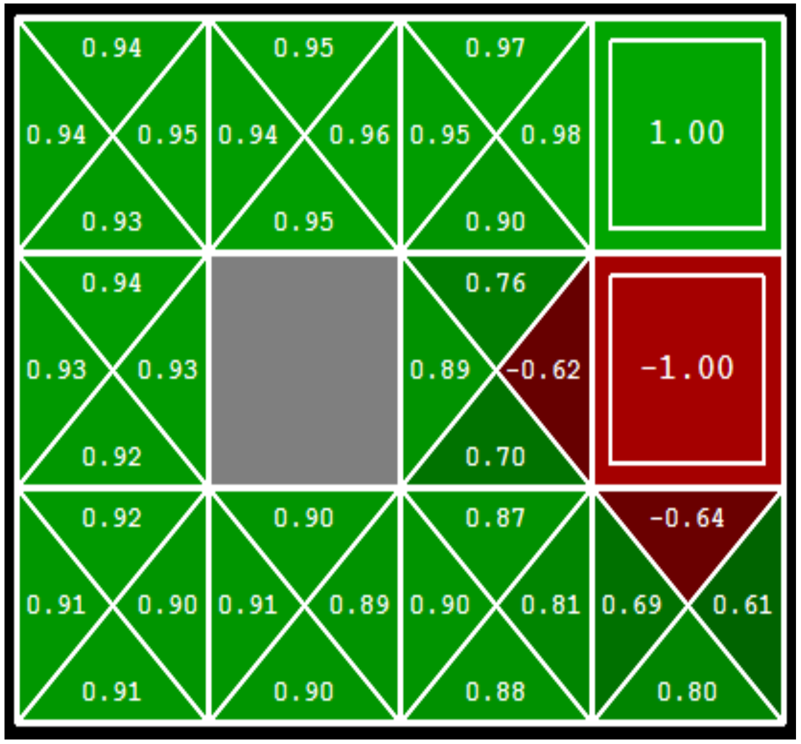
Computing actions from Q-values

Let's imagine we have the optimal q-values:

How should we act?

Completely trivial to decide!

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$



Important lesson: actions are easier to select from q-values than values!

Policy iteration

Alternative approach for optimal values:

- Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
- Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
- Repeat steps until policy converges

This is policy iteration

- It's still optimal!
- Can converge (much) faster under some conditions

Online methods

Online methods compute optimal action from current state

Expand tree up to some horizon

States reachable from the current state is typically small compared to full state space

Heuristics and branch-and-bound techniques allow search space to be pruned

Monte Carlo methods provide approximate solutions

Problem set question

We assume there is a MDP which has a finite number of actions and states.

(a) What's the condition that can make this MDP guaranteed to converge? Why?

(b) Do the converged values change based on different initial values? Why?

(c) If the values in value iteration have just converged, is the policy converged as well at that time? Why?

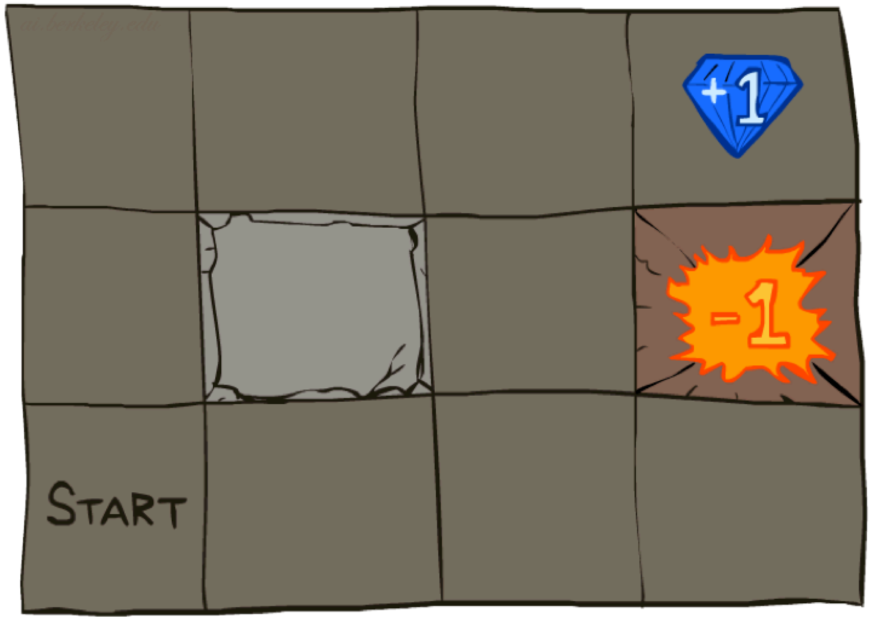
(d) If the policy in value iteration have just converged, are the values converged as well at that time? Why?

(e) If two MDPs have the same actions and states, except the discount factor and they both converge, will they have the same policy?

Reinforcement learning

Reinforcement Learning

~~We know the reward function~~



~~We know the probabilities of moving in each direction when an action is executed~~

Reinforcement Learning

Still assume a Markov decision process (MDP):

- A set of states $s \in S$
- A set of actions (per state) A
- A model T(s,a,s')
- A reward function R(s,a,s')

Still looking for a policy $\pi(s)$

New twist: don't know T or R

- I.e. we don't know which states are good or what the actions do
- Must actually try actions and states out to learn

Model-based RL

1. estimate T, R by averaging experiences

2. solve for policy in MDP (e.g., value iteration)

a. choose an exploration policy
– policy that enables agent to explore all relevant states

b. follow policy for a while

c. estimate T and R

$N_{s,a,s'} \equiv$ Number of times agent reached *s'* by taking *a* from *s*

$R_{s,a,s'} \equiv$ Set of rewards obtained when reaching *s'* by taking *a* from *s*

$$T(s,a,s') \approx \frac{N_{s,a,s'}}{\sum_{s'} N_{s,a,s'}} \qquad R(s,a,s') \approx \frac{1}{N_{s,a,s'}} R_{s,a,s'}$$

Model-based vs Model-free learning

Goal: Compute expected age of students in this class

Known P(A)

$$E[A] = \sum_a P(a) \cdot a \qquad = 0.35 \times 20 + \ldots$$

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): "Model Based"

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Why does this work? Because eventually you learn the right model.

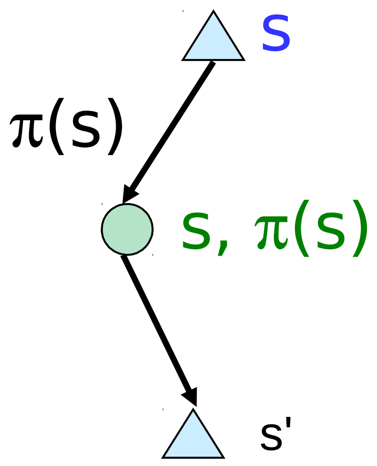
Unknown P(A): "Model Free"

$$E[A] \approx \frac{1}{N} \sum_i a_i$$

Why does this work? Because samples appear with the right frequencies.

TD Value Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often



- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average (incremental mean)

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

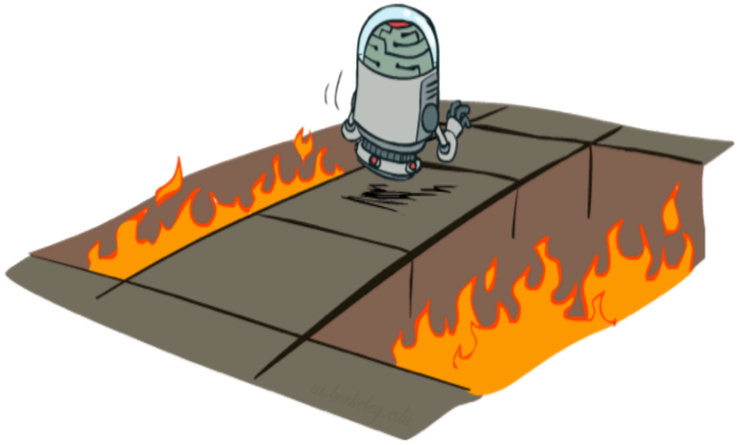
Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Active Reinforcement Learning

Full reinforcement learning: generate optimal policies (like value iteration)

- You don't know the transitions T(s,a,s')
- You don't know the rewards R(s,a,s')
- You choose the actions now
- Goal: learn the optimal policy / values



In this case:

- Learner makes choices!
- Fundamental tradeoff: exploration vs. exploitation
- This is NOT offline planning! You actually take actions in the world and find out what happens…

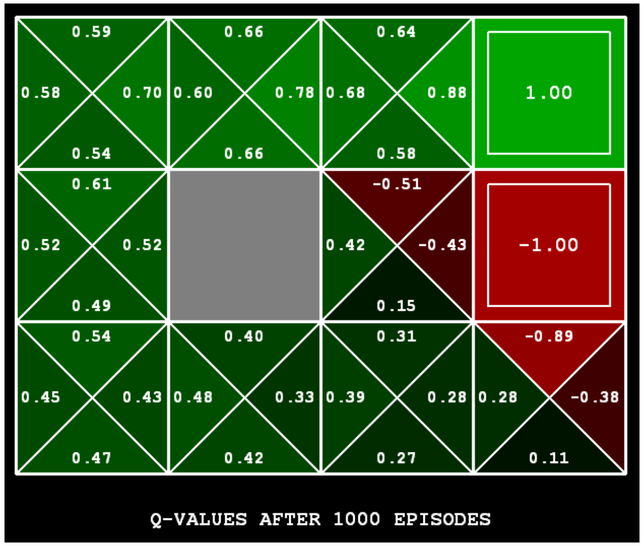
Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$



  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[sample\right]$$

Q-Learning: properties

Q-learning converges to optimal Q-values if:

1. it explores every s, a, s' transition sufficiently often

2. the learning rate approaches zero (eventually)

Key insight: Q-value estimates converge even if experience is obtained using a suboptimal policy.

This is called off-policy learning

How to explore?

Several schemes for forcing exploration

- Simplest: random actions (ε-greedy)
  - Every time step, flip a coin
  - With (small) probability ε, act randomly
  - With (large) probability 1-ε, act on current policy

- Problems with random actions?
  - You do eventually explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

Exploration functions

When to explore?

Random actions: explore a fixed amount

Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration function

Takes a value estimate *u* and a visit count *n*, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

Regular Q-Update: $Q(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} Q(s',a')$

Modified Q-Update: $Q(s,a) \leftarrow_{\alpha} R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), N(s',a'))$

Note: this propagates the “bonus” back to states that lead to unknown states as well!

Generalization: Linear value functions

Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

Advantage: our experience is summed up in a few powerful numbers

Disadvantage: states may share features but actually be very different in value!

Policy search

Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best

- E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
- Q-learning's priority: get Q-values close (modeling)
- Action selection priority: get ordering of Q-values right (prediction)
- We'll see this distinction between modeling and prediction again later in the course

Solution: learn policies that maximize rewards, not the values that predict them

Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

Problem set question

1. When learning with ε-greedy action selection, is it a good idea to decrease ε to 0 with time? Why or why not?
2. When using features to represent the Q-function is it guaranteed that the feature-based Q-learning finds the same optimal Q^* as would be found when using a tabular representation for the Q-function?
3. Does the temporal difference learning of optimal utility values (U) require knowledge of the transition probability tables? Why or why not?
4. Why is temporal difference (TD) learning of Q-values (Q-learning) superior to TD learning of values?

Bayes nets

Probabilistic inference

Probabilistic inference: compute a desired probability from other known probabilities (e.g. conditional from joint)

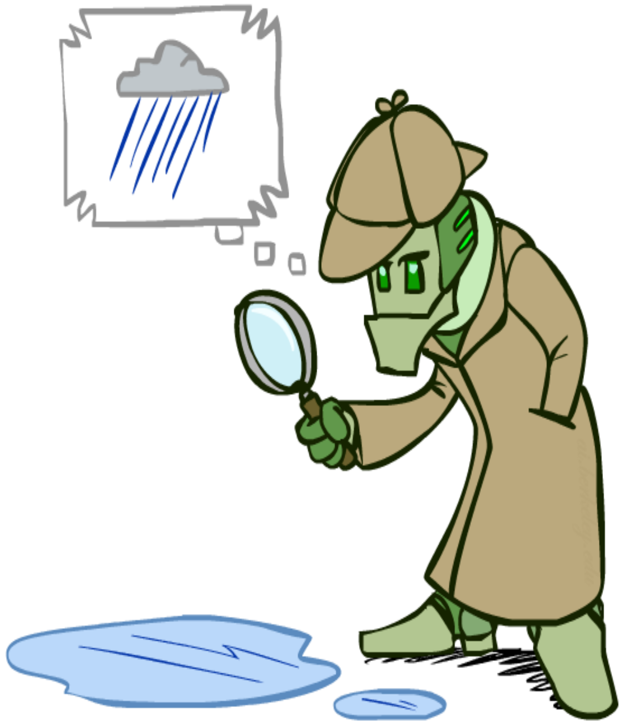
We generally compute conditional probabilities

- P(on time | no reported accidents) = 0.90
- These represent the agent's *beliefs* given the evidence

Probabilities change with new evidence:

- P(on time | no accidents, 5 a.m.) = 0.95
- P(on time | no accidents, 5 a.m., raining) = 0.80
- Observing new evidence causes *beliefs to be updated*

We'll cover inference techniques next class!

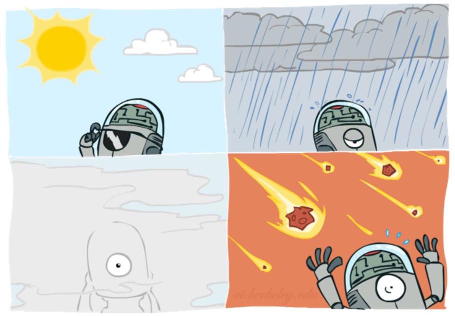


Graphical model notation

Nodes: variables (with domains)

- Can be assigned (observed) or unassigned (unobserved)

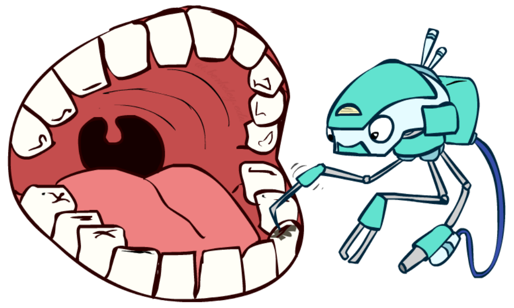
$$\begin{tikzcd} \text{Weather} \end{tikzcd}$$



Arcs: interactions

- Similar to CSP constraints
- Indicate “direct influence” between variables
- Formally: encode conditional independence (more later)

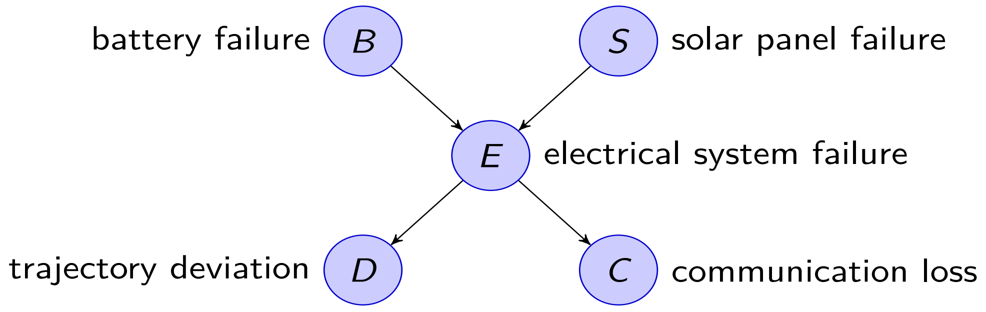
$$\begin{tikzcd} & \text{Cavity} \arrow[dl] \arrow[dr] & \\ \text{Toothache} & & \text{Catch} \end{tikzcd}$$



Bayes' net semantics

- A directed, acyclic graph, one node per random variable
- A conditional probability table (CPT) for each node
  - A collection of distributions over X, one for each combination of parents' values $P(X|a_1 \dots a_n)$
- Bayes' nets implicitly encode joint distributions
  - As a product of local conditional distributions
  - To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:

$$P(x_1, x_2, \dots x_n) = \prod_{i=1}^{n} P(x_i|\mathit{parents}(X_i))$$



Probabilities in Bayes' nets

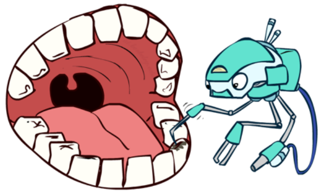
Bayes' nets **implicitly** encode joint distributions

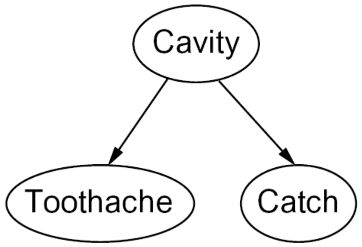
As a product of local conditional distributions

To see what probability a BN gives to a full assignment, multiply all the relevant conditionals together:

$$P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i|\mathit{parents}(X_i))$$

Example:

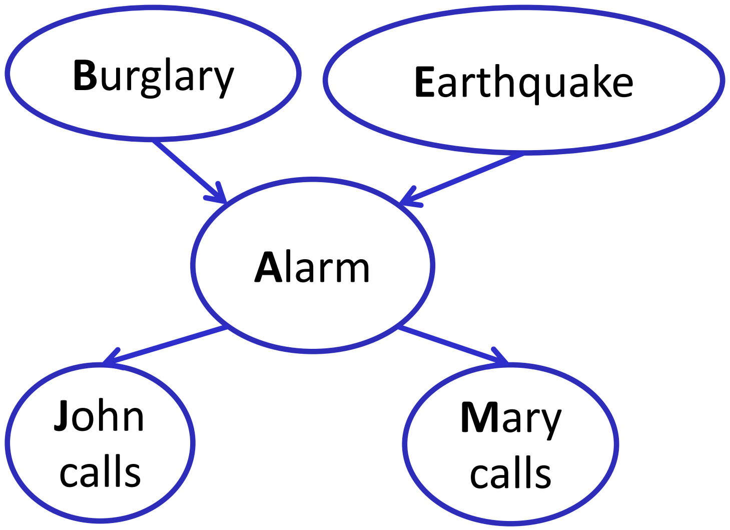




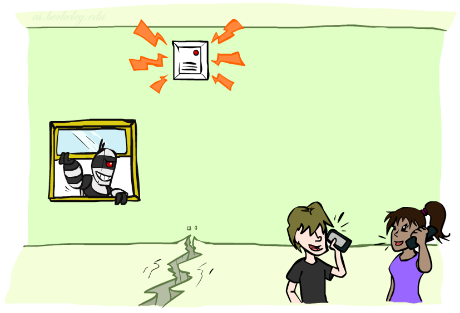
$P(+\mathit{cavity}, +\mathit{catch}, -\mathit{toothache})$

Example: Alarm network

B	P(B)
+b	0.001
-b	0.999



E	P(E)
+e	0.002
-e	0.998



A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

Size of a Bayes' net

- How big is a joint distribution over N Boolean variables?

  2^N

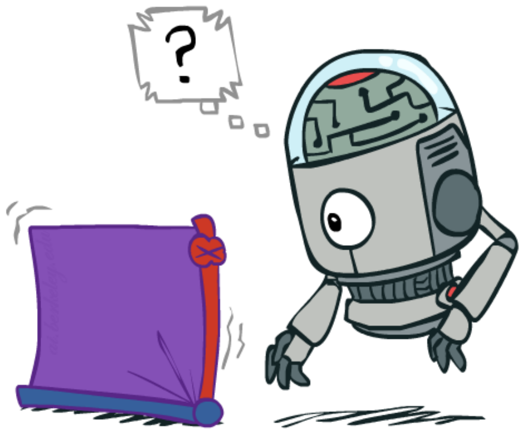
- How big is an N-node net if nodes have up to k parents?

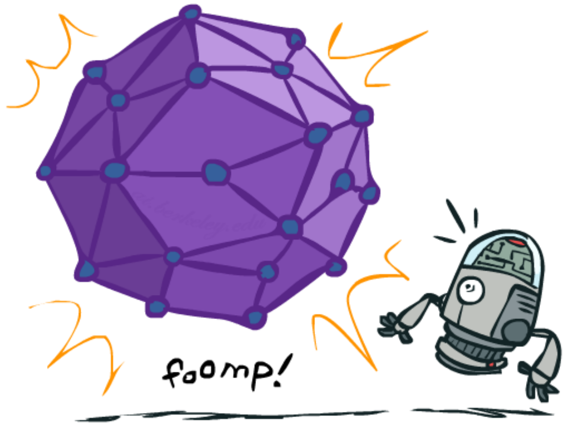
  $O(N * 2^{k+1})$

- Both give you the power to calculate

$$P(X_1, X_2, \ldots X_n)$$

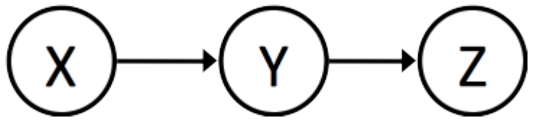
- BNs: Huge space savings!
- Also easier to elicit local CPTs
- Also faster to answer queries (coming)





Independence in a Bayes' net

- Important question about a BN:
  - Are two nodes independent given certain evidence?
  - If yes, can prove using algebra (tedious in general)
  - If no, can prove with a counter example
  - Example:

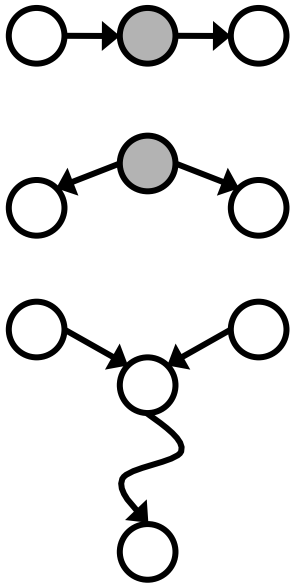


  - Question: are X and Z necessarily independent?
    - Answer: no. Example: low pressure causes rain, which causes traffic.
    - X can influence Z, Z can influence X (via Y)
    - Addendum: they *could* be independent: how?

Independence concepts: General case

Two sets of nodes, A and B, are conditionally independent given node set C are called d-separated in the Bayes net if for any path between A and B:

- The path contains a chain of nodes, $A \rightarrow X \rightarrow B$, such that X is in C
- The path contains a fork, $A \leftarrow X \rightarrow B$, such that X is in C
- The path contains a v-structure, $A \rightarrow X \leftarrow B$, such that X is *not* in C and no descendant of X is in C

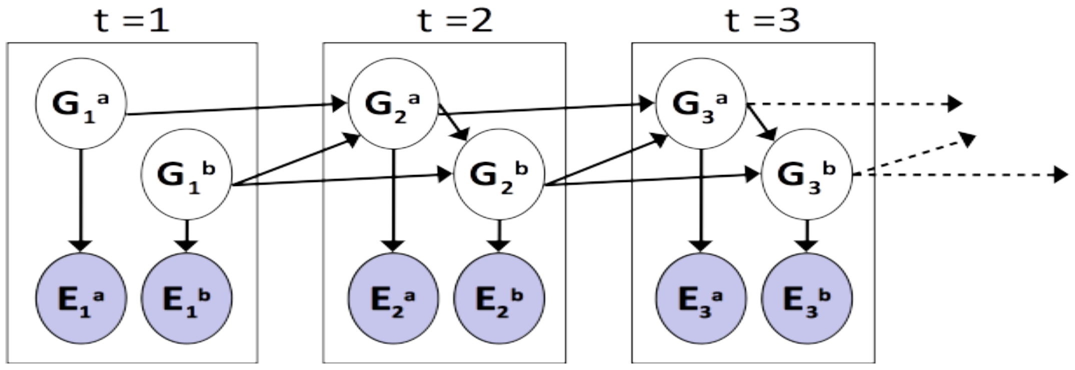


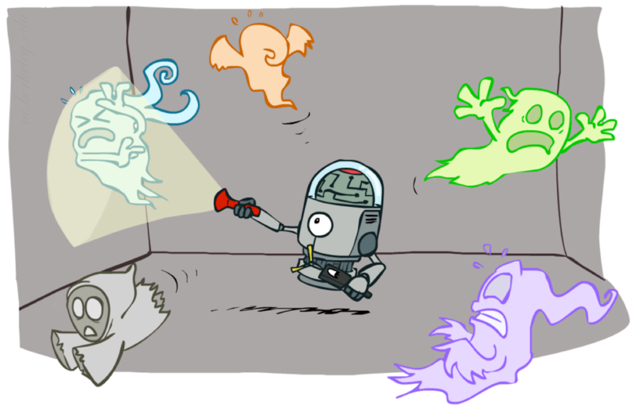
Markov blanket: the parents, the children and the parents of the children

- A node is independent of all other nodes in the graph given its Markov blanket

Dynamic Bayes Nets (DBNs)

- We want to track multiple variables over time, using multiple sources of evidence
- Idea: Repeat a fixed Bayes net structure at each time
- Variables from time *t* can condition on those from *t-1*





- Dynamic Bayes nets are a generalization of HMMs

Inference

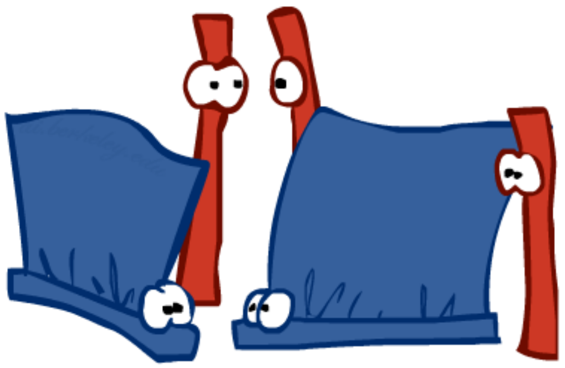
- Inference: calculating some useful quantity from a joint probability distribution

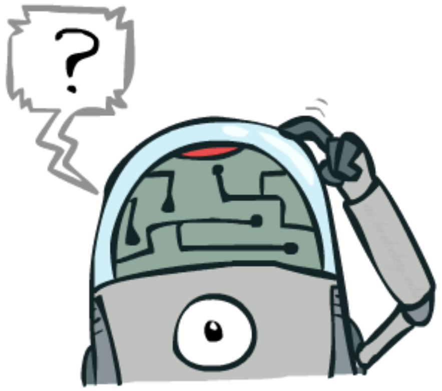
- Examples:
  - Posterior probability

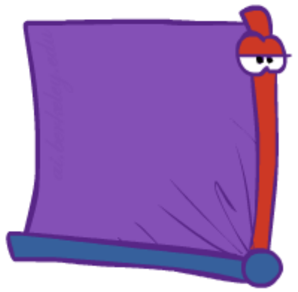
$$P(Q|E_1 = e_1, \ldots E_k = e_k)$$

  - Most likely explanation:

$$\text{argmax}_q \; P(Q = q|E_1 = e_1 \ldots)$$

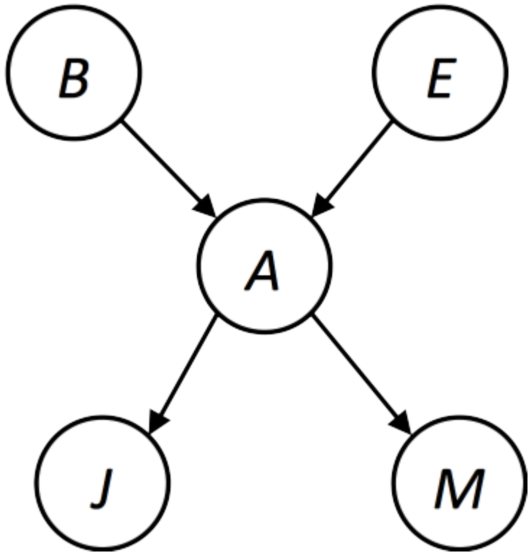




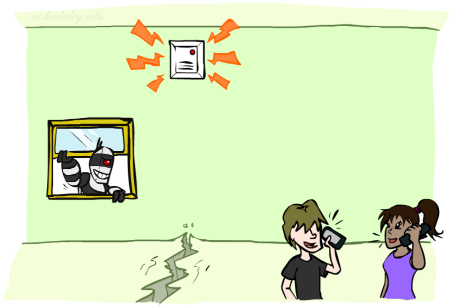


Burglar alarm example

B	P(B)
+b	0.001
-b	0.999



E	P(E)
+e	0.002
-e	0.998



A	J	P(J|A)
+a	+j	0.9
+a	-j	0.1
-a	+j	0.05
-a	-j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	-m	0.3
-a	+m	0.01
-a	-m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	-a	0.05
+b	-e	+a	0.94
+b	-e	-a	0.06
-b	+e	+a	0.29
-b	+e	-a	0.71
-b	-e	+a	0.001
-b	-e	-a	0.999

$$P(+b, -e, +a, -j, +m) =$$
$$P(+b)P(-e)P(+a| + b, -e)P(-j| + a)P(+m| + a) =$$
$$0.001 \times 0.998 \times 0.94 \times 0.1 \times 0.7$$

Inference by enumeration

General case:

Evidence variables: $E_1 \dots E_k = e_1 \dots e_k$
Query* variable: Q
Hidden variables: $H_1 \dots H_r$

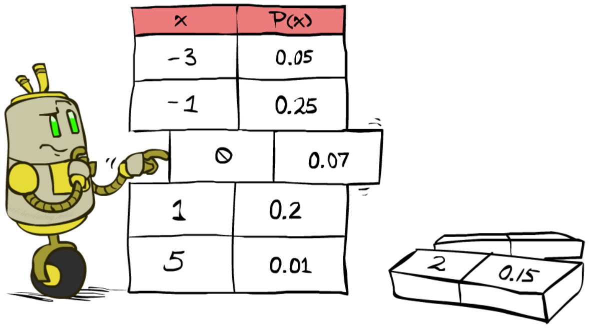
$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\}$ $X_1, X_2, \dots X_n$ *All variables*

- We want:

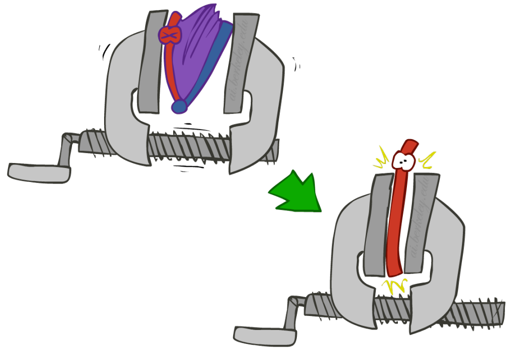
** Works fine with multiple query variables, too*

$$P(Q|e_1 \dots e_k)$$

- Step 1: Select the entries consistent with the evidence



- Step 2: Sum out H to get joint of query and evidence



$$P(Q, e_1 \dots e_k) = \sum_{h_1 \dots h_r} \underbrace{P(Q, h_1 \dots h_r, e_1 \dots e_k)}_{X_1, X_2, \dots X_n}$$

- Step 3: Normalize

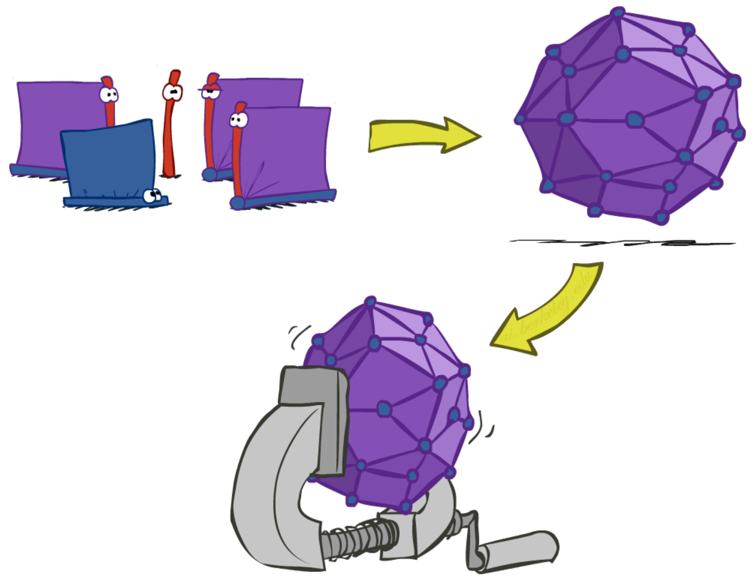
$$\times \frac{1}{Z}$$

$$Z = \sum_q P(Q, e_1 \cdots e_k)$$

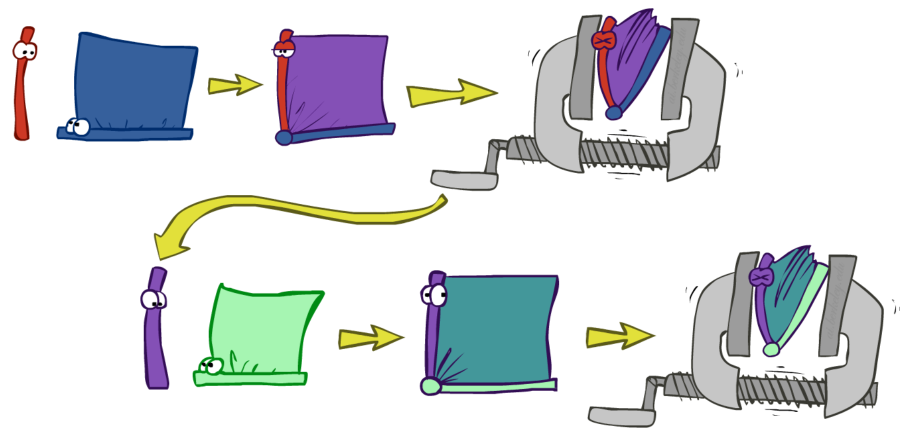
$$P(Q|e_1 \cdots e_k) = \frac{1}{Z} P(Q, e_1 \cdots e_k)$$

Inference by enumeration vs. variable elimination

- Why is inference by enumeration so slow?
  - You join up the whole joint distribution before you sum out the hidden variables

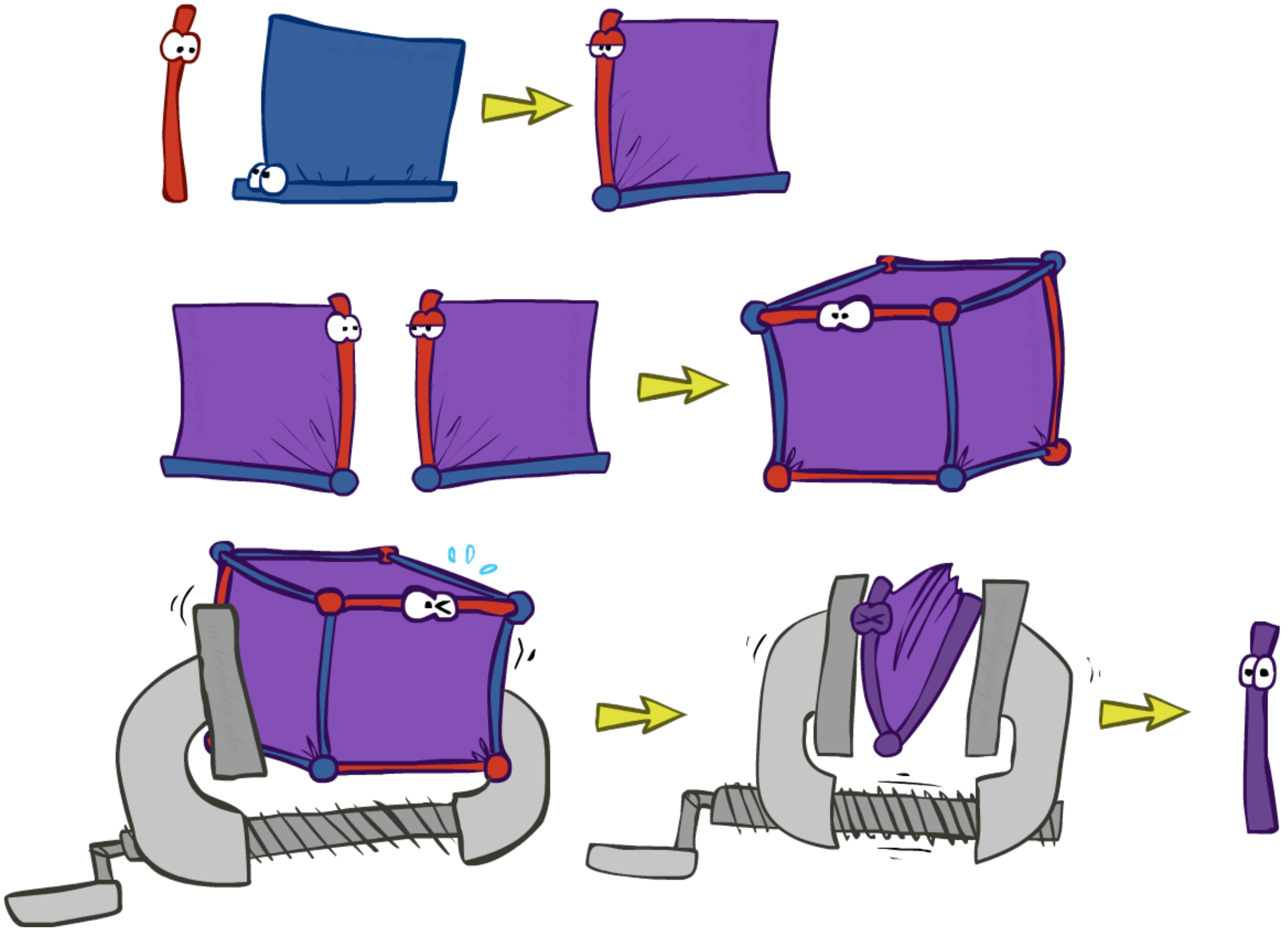


- Idea: interleave joining and marginalizing!
  - Called "Variable Elimination"
  - Sum right-to-left, storing intermediate results (*factors*) to avoid recomputation
  - Still NP-hard, but usually much faster than inference

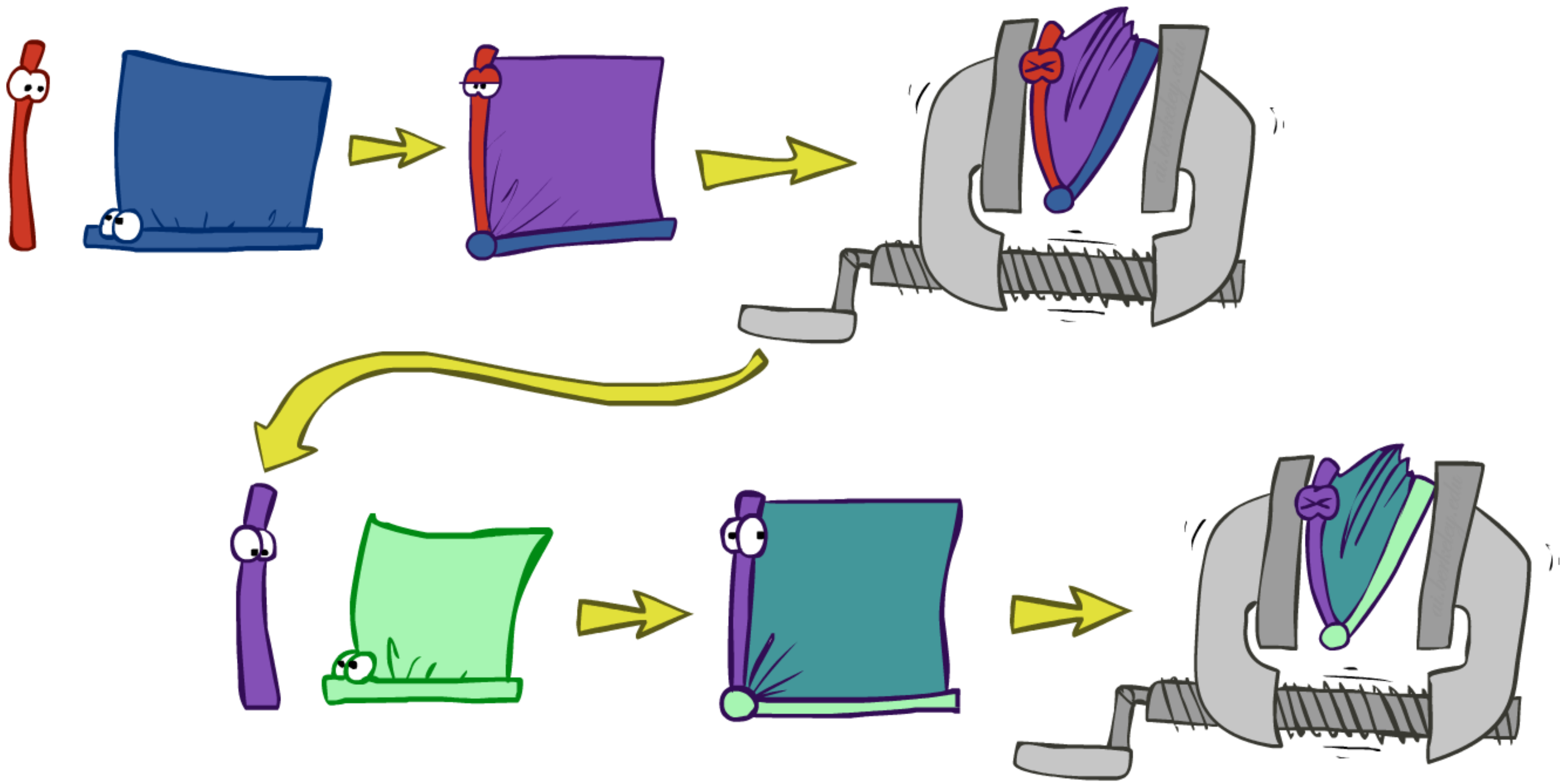


  - First we'll need some new notation: factors

Thus far: Multiple join, multiple eliminate (= inference by enumeration)



Marginalizing early (= variable elimination)



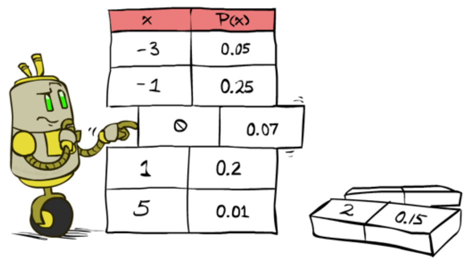
General variable elimination

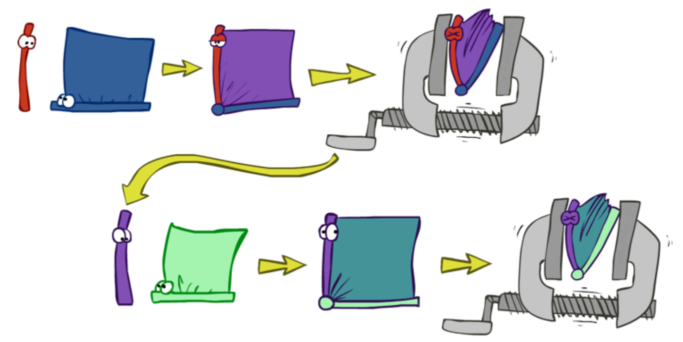
- Query: $P(Q|E_1 = e_1, \dots E_k = e_k)$

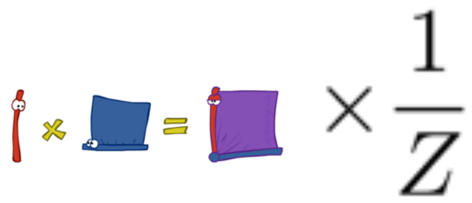
- Start with initial factors:
  - Local CPTs (but instantiated by evidence)

- While there are still hidden variables (not Q or evidence):
  - Pick a hidden variable H
  - Join all factors mentioning H
  - Eliminate (sum out) H

- Join all remaining factors and normalize







VE: Computation and space complexity

- The computational and space complexity of variable elimination is determined by the largest factor
- The elimination ordering can greatly affect the size of the largest factor
  - E.g., 2^n vs. 2
- Does there always exist an ordering that only results in small factors?
  - No!

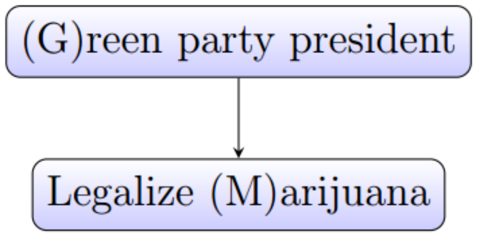
Approximate inference through sampling

```
function DirectSample($B$)
    $X_{1:n} \leftarrow$ a topological sort of nodes in $B$
    for $i \leftarrow 1$ to $n$
        $x_i \leftarrow$ a random sample from $P(X_i \mid \mathrm{pa}_{x_i})$
    return $x_{1:n}$
```

Direct sampling in a Bayes Net

Problem set question

It’s election season, and the chosen president may or may not be the Green Party candidate. Pundits believe that Green Party presidents are more likely to legalize marijuana than candidates from other parties, but legalization could occur under any administration. Armed with the power of probability, the analysts model the situation with the Bayes Net below.



	+g	-g
P(g)	0.25	0.75

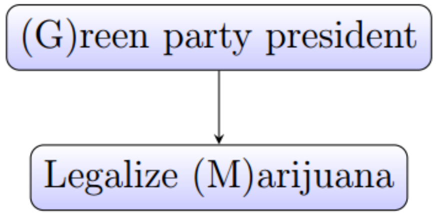
	$P(+m\|g)$	$P(-m\|g)$
+g	0.8	0.2
-g	0.1	0.9

1. What is the marginal probability that marijuana is legalized P (+m)?

2. News agencies air 24/7 coverage of the recent legalization of marijuana (+m), but you can’t seem to find out who won the election. What is the conditional probability P (+g| + m) that a Green Party president was elected?

Problem set question

It’s election season, and the chosen president may or may not be the Green Party candidate. Pundits believe that Green Party presidents are more likely to legalize marijuana than candidates from other parties, but legalization could occur under any administration. Armed with the power of probability, the analysts model the situation with the Bayes Net below.



	+g	-g
P(g)	0.25	0.75

	$P(+m\|g)$	$P(-m\|g)$
+g	0.8	0.2
-g	0.1	0.9

1. What is the marginal probability that marijuana is legalized P (+m)?

$$P(+m) = P(+m|+g)P(+g) + P(+m|-g)P(-g)$$

$$P(+m) = 0.8 * 0.25 + 0.1 * 0.75$$

$$P(+m) = 0.275$$

2. News agencies air 24/7 coverage of the recent legalization of marijuana (+m), but you can’t seem to find out who won the election. What is the conditional probability P (+g| + m) that a Green Party president was elected?

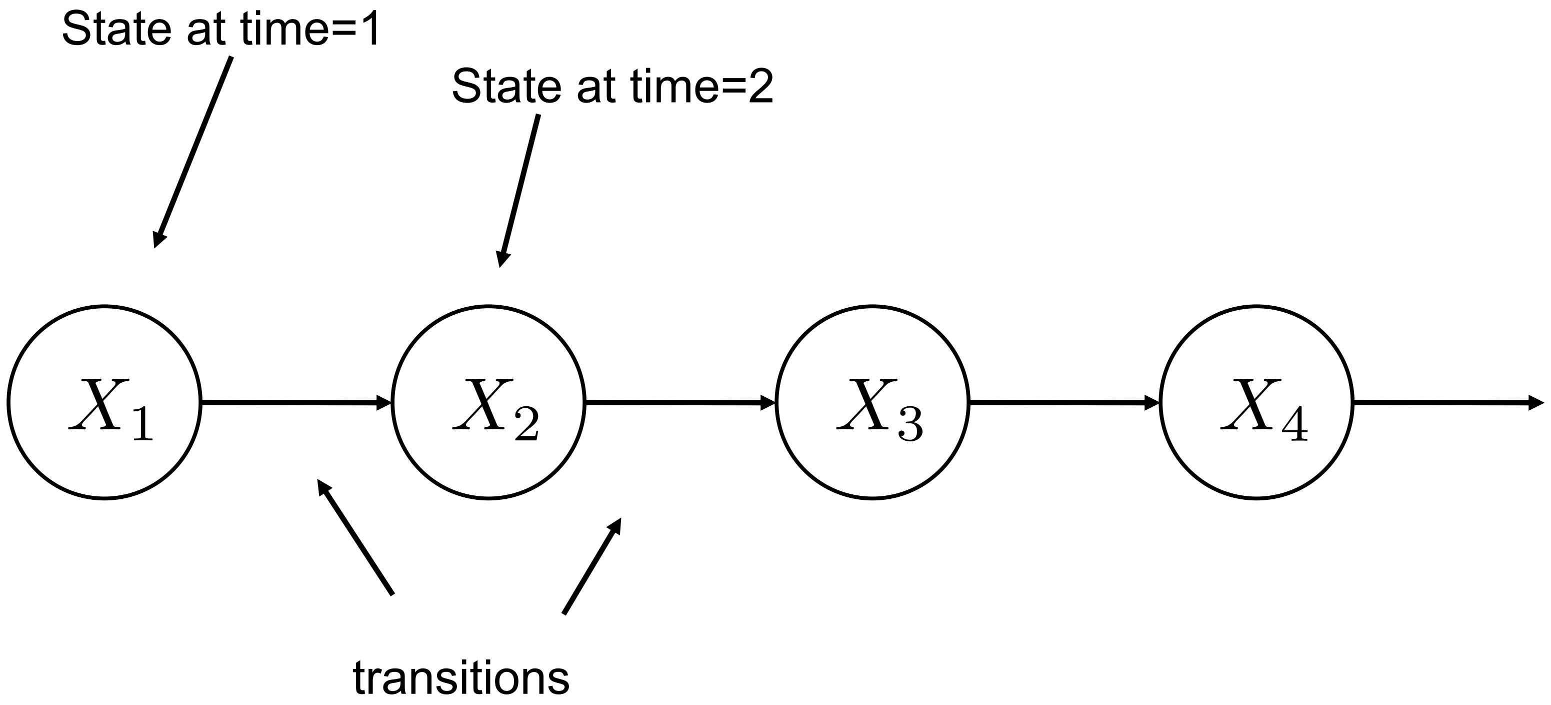
$$P(+g|+m) = \frac{P(+m|+g)P(+g)}{P(+m)}$$

$$P(+g|+m) = \frac{0.8 * 0.25}{0.275}$$

$$P(+g|+m) = 0.727$$

(Hidden) Markov models

Markov Processes



Since this is a Markov process, we assume transitions are Markov:

Process model: $P(X_t|X_{t-1}) = P(X_t|X_{t-1}, \ldots, X_1)$

Markov assumption: $X_t \perp\!\!\!\perp X_{t-2}|X_{t-1}$

Markov Processes

$$\begin{tikzcd}
X_1 \arrow[r] & X_2 \arrow[r] & X_3 \arrow[r] & X_4 \arrow[r] & {}
\end{tikzcd}$$

How do we calculate: $P(X_1, X_2, X_3, X_4) = ?$

$P(X_1, X_2, X_3, X_4) = P(X_1)P(X_2|X_1)P(X$ Process model $X_2, X_1)$

$$P(X_1, X_2, X_3, X_4) = P(X_1)P(X_2|X_1)P(X_3|X_2)P(X_4|X_3)$$

In general: $P(X_1, X_2, \ldots, X_T) = P(X_1) \prod_{t=1}^{T-1} \boxed{P(X_{t+1}|X_t)}$

Simulating dynamics forward

$$\begin{tikzcd} X_1 \arrow[r] & X_2 \arrow[r] & X_3 \arrow[r] & X_4 \arrow[r] & {} \end{tikzcd}$$

Joint distribution: $P(X_1, X_2, \ldots, X_T) = P(X_1) \prod_{t=1}^{T-1} P(X_{t+1}|X_t)$

But, suppose we want to predict the state at time T, given a prior distribution at time 1?

$$P(X_2) = \sum_{X_1} P(X_1)P(X_2|X_1)$$

$$P(X_3) = \sum_{X_2} P(X_2)P(X_3|X_2)$$

…

$$P(X_T) = \sum_{X_{T-1}} P(X_{T-1})P(X_T|X_{T-1})$$

Hidden Markov Models (HMMs)

$$\begin{tikzcd}
X_1 \arrow[r] \arrow[d] & X_2 \arrow[r] \arrow[d] & X_3 \arrow[r] \arrow[d] & X_4 \arrow[r] \arrow[d] & {} \\
E_1 & E_2 & E_3 & E_4 &
\end{tikzcd}$$

Called an “emission”

State, X_t , is assumed to be unobserved

However, you get to make one observation, E_t , on each timestep.

HMM Filtering

Given a prior distribution, $P(X_1)$, and a series of observations, $E_1, \dots, E_T$, calculate the posterior distribution: $P(X_t|E_1, \dots, E_T)$

Two steps:

$$\begin{tikzcd} \boxed{P(X_t|E_{1:t})} \arrow[r, "\text{Process update}"'] & \boxed{P(X_{t+1}|E_{1:t})} \arrow[r, "\text{Observation update}"'] & \boxed{P(X_{t+1}|E_{1:t+1})} \end{tikzcd}$$

Process update

$$\begin{tikzcd}
B(X_t) & B'(X_t) \\
\boxed{P(X_t|e_{1:t})} \arrow[r] & \boxed{P(X_{t+1}|e_{1:t})}
\end{tikzcd}$$

$$P(X_{t+1}|e_{1:t}) = \sum_{X_t} P(X_{t+1}|X_t, e_{1:t})P(X_t|e_{1:t})$$

$$B'(X_{t+1}) = \sum_{X_t} P(X_{t+1}|X_t, e_{1:t})B(X_t)$$

$$\uparrow$$

This is just forward simulation of the Markov Model

Observation update

$$\begin{tikzcd} B'(X_t) & B(X_{t+1}) \\ \boxed{P(X_{t+1}|e_{1:t})} \arrow[r] & \boxed{P(X_{t+1}|e_{1:t+1})} \end{tikzcd}$$

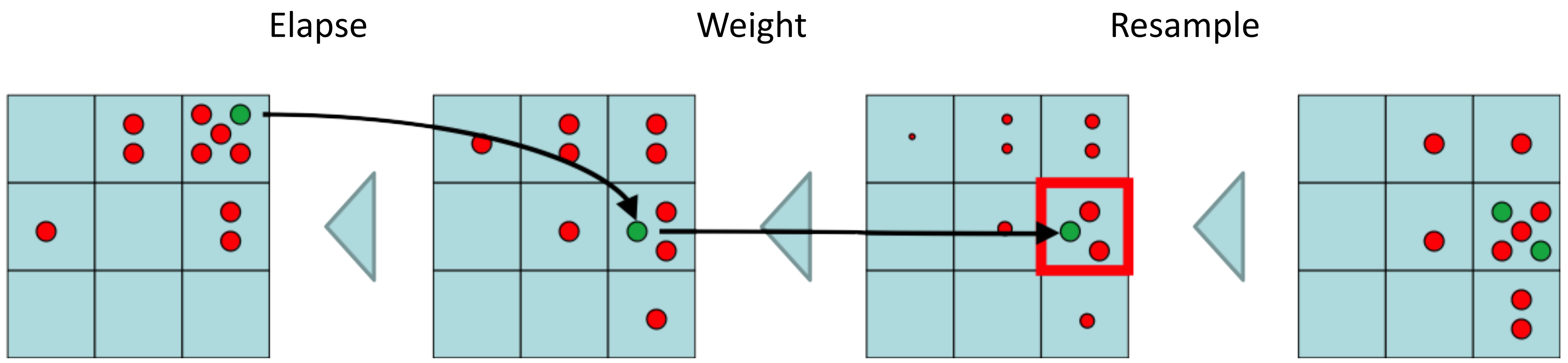
$$P(X_{t+1}|e_{1:t+1}) = \eta P(e_{t+1}|X_{t+1}) P(X_{t+1}|e_{1:t})$$

$$B(X_{t+1}) = \eta P(e_{t+1}|X_{t+1}) B'(X_{t+1})$$

Where $\eta = \dfrac{1}{P(e_{t+1})}$ is a normalization factor

Recap: Particle Filtering

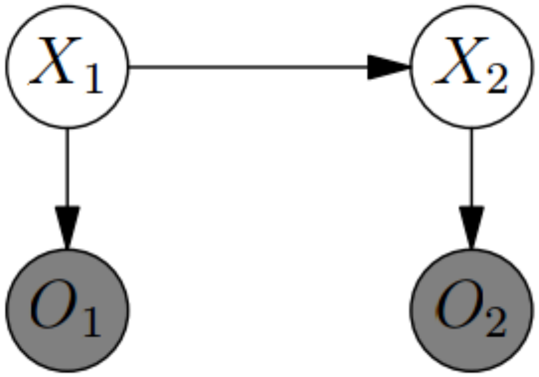
- Particles: track samples of states rather than an explicit distribution



Particles:	Particles:	Particles:	(New) Particles:
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,3)	(2,3) w=.2	(2,2)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(3,2)	(3,1)	(3,1) w=.4	(2,3)
(3,3)	(3,3)	(3,3) w=.4	(3,3)
(3,2)	(3,2)	(3,2) w=.9	(3,2)
(1,2)	(1,3)	(1,3) w=.1	(1,3)
(3,3)	(2,3)	(2,3) w=.2	(2,3)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,2)	(2,2) w=.4	(3,2)

Problem set question

Consider the following Hidden Markov Model:



$$\begin{tikzcd} X_1 \arrow[r] \arrow[d] & X_2 \arrow[d] \\ O_1 & O_2 \end{tikzcd}$$

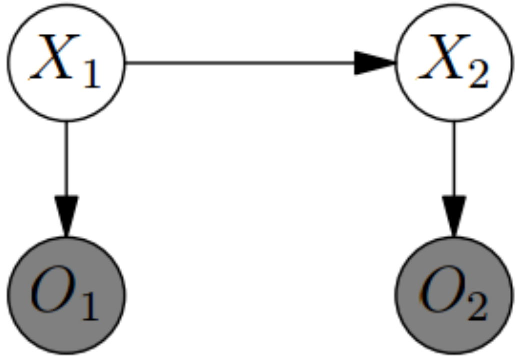
Suppose that O_1 = A and O_2 = B is observed. Use the Forward algorithm to compute the probability distribution $Pr(X_2, O_1 = A, O_2 = B)$. Show your work.

X_1	$P(X_1)$
0	0.25
1	0.75

X_t	X_{t+1}	$Pr(X_{t+1}\|X_t)$
0	0	0.3
0	1	0.7
1	0	0.6
1	1	0.4

X_t	O_t	$Pr(O_t\|X_t)$
0	A	0.8
0	B	0.2
1	A	0.5
1	B	0.5

Problem set question



Suppose that O_1 = A and O_2 = B is observed. Use the Forward algorithm to compute the probability distribution $Pr(X_2, O_1 = A, O_2 = B)$. Show your work.

X_1	$P(X_1)$
0	0.25
1	0.75

X_t	X_{t+1}	$Pr(X_{t+1}\|X_t)$
0	0	0.3
0	1	0.7
1	0	0.6
1	1	0.4

X_t	O_t	$Pr(O_t\|X_t)$
0	A	0.8
0	B	0.2
1	A	0.5
1	B	0.5

Solution:
Forward algorithm: $P(x_t, e_{1:t}) = P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}, e_{1:t-1})$

$$\boldsymbol{P}(X_2, O_1 = A, O_2 = B) = \boldsymbol{P}(O_2 = B|X_2) \sum_{X_1} \boldsymbol{P}(X_2|X_1) \boldsymbol{P}(X_1, O_1 = A)$$

$$\boldsymbol{P}(X_2, O_1 = A, O_2 = B) = \eta * < 0.5, 0.2 > *((< 0.4, 0.6 > *0.5 * 0.75)_{X_1=1} + (< 0.7, 0.3 > *0.8 * 0.25)_{X_1=0})$$

$$\boldsymbol{P}(X_2, O_1 = A, O_2 = B) = \eta * < 0.5 * (0.4 * 0.5 * 0.75 + 0.7 * 0.8 * 0.25), 0.2 * (0.6 * 0.5 * 0.75 + 0.3 * 0.8 * 0.25) >$$

$$\boldsymbol{P}(X_2, O_1 = A, O_2 = B) = \eta * < 0.145, 0.057 >; \eta = 4.95, \boldsymbol{P} =< 0.71775, 0.28215 >$$

Classification

Supervised learning

Given: *Training set* $\{(x_i, y_i) \mid i = 1 \ldots N\}$, given a labeled set of input-output pairs $D = \{(x_i, y_i)\}_i$

Find: A good approximation to $f : X \rightarrow Y$ Function approximation

Examples: what are X and Y ?

Spam Detection – Map email to {Spam, Not Spam} Binary Classification

Digit recognition – Map pixels to {0,1,2,3,4,5,6,7,8,9} Multiclass Classification

Stock Prediction – Map new, historic prices, etc. to (the real numbers) Regression

Probabilistic Classification

Want a probability distribution over possible labels, given the input vector x and training set D by $P(y|x,D)$

In general, this represents a vector of length C

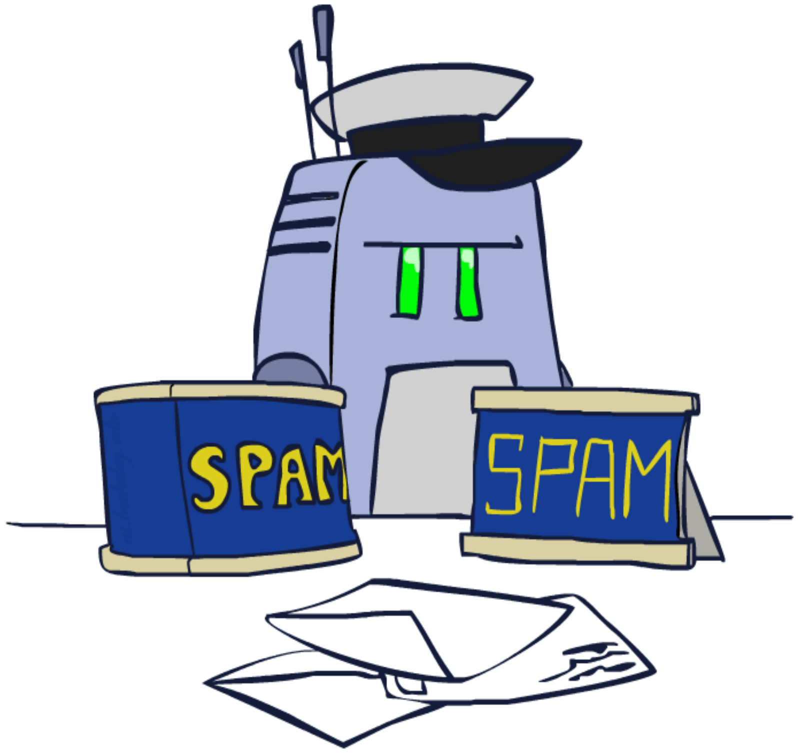
Calculate a "best guess": $\hat{y} = \hat{f}(x) = \operatorname{argmax}_{c=1}^{C} P(y = c \mid x, D)$

This corresponds to the most probable class label (the mode of the distribution)

Model-Based Classification

- Model-based approach
  - Build a model (e.g. Bayes' net) where both the label and features are random variables
  - Instantiate any observed features
  - Query for the distribution of the label conditioned on the features

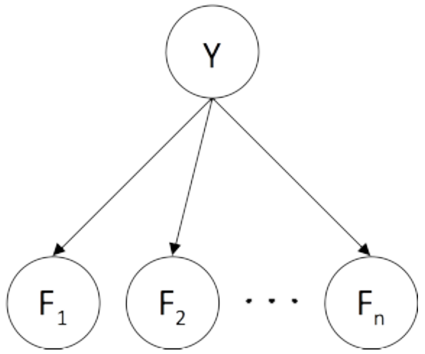
- Challenges
  - What structure should the BN have?
  - How should we learn its parameters?



General Naïve Bayes

- A general Naive Bayes model:

$$\underbrace{P(\mathsf{Y},\mathsf{F}_1\ldots\mathsf{F}_n)}_{|Y| \times |F|^n \text{ values}} = \overbrace{P(\mathsf{Y})}^{|Y| \text{ parameters}} \underbrace{\prod_i P(\mathsf{F}_i|\mathsf{Y})}_{n \times |F| \times |Y| \text{ parameters}}$$



- We only have to specify how each feature depends on the class
- Total number of parameters is *linear* in n
- Model is very simplistic, but often works anyway

General Naïve Bayes

- What do we need in order to use Naïve Bayes?
  - Inference method (we just saw this part)
    - Start with a bunch of probabilities: $P(Y)$ and the $P(F_i|Y)$ tables
    - Use standard inference to compute $P(Y|F_1 \ldots F_n)$
    - Nothing new here
  - Estimates of local conditional probability tables
    - $P(Y)$, the prior over labels
    - $P(F_i|Y)$ for each feature (evidence variable)
    - These probabilities are collectively called the *parameters* of the model and denoted by θ
    - Up until now, we assumed these appeared by magic, but…
    - …they typically come from training data counts: we'll look at this soon

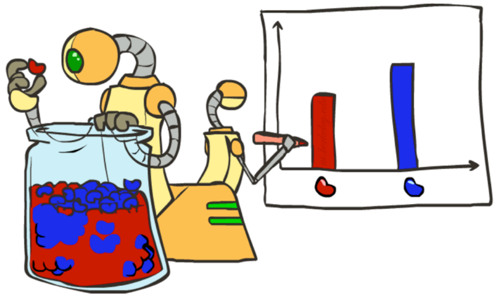
Parameter Estimation

- Estimating the distribution of a random variable
- *Elicitation:* ask a human (why is this hard?)
- *Empirically:* use training data (learning!)
  - E.g.: for each outcome x, look at the *empirical rate* of that value:

$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

r r b

$$P_{\text{ML}}(\text{r}) = 2/3$$



  - This is the estimate that maximizes the *likelihood of the data*

$$L(x, \theta) = \prod_i P_\theta(x_i)$$

Maximum Likelihood?

- Relative frequencies are the maximum likelihood estimates

$$\begin{aligned}\theta_{ML} &= \underset{\theta}{\text{arg max}}\, P(\mathbf{X}|\theta) \\ &= \underset{\theta}{\text{arg max}} \prod_i P_\theta(X_i)\end{aligned}$$

$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

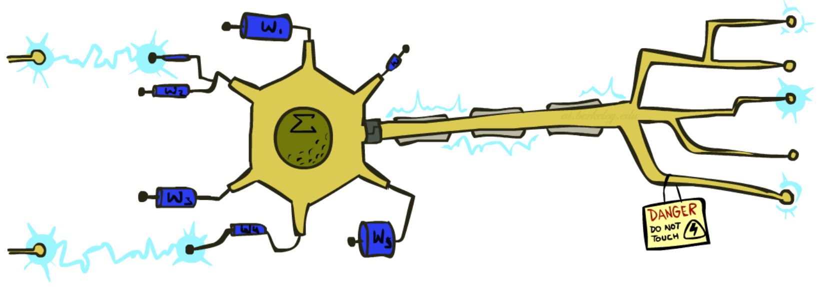
- Another option is to consider the most likely parameter value given the data

$$\begin{aligned}\theta_{MAP} &= \underset{\theta}{\text{arg max}}\, P(\theta|\mathbf{X}) \\ &= \underset{\theta}{\text{arg max}}\, P(\mathbf{X}|\theta)P(\theta)/P(\mathbf{X}) \\ &= \underset{\theta}{\text{arg max}}\, P(\mathbf{X}|\theta)P(\theta)\end{aligned}$$

$\Rightarrow$????

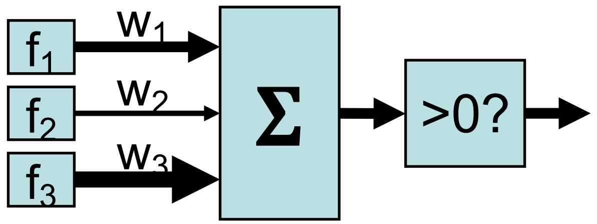
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

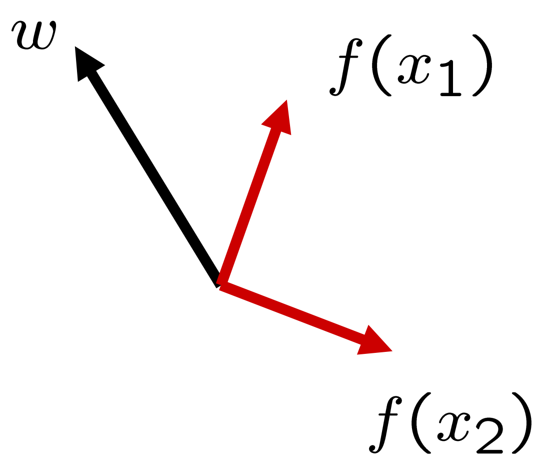
- If the activation is:
  - Positive, output +1
  - Negative, output -1



Weights

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples

```
# free      : 4
YOUR_NAME   :-1
MISSPELLED  : 1
FROM_FRIEND :-3
...
```



```
# free      : 2
YOUR_NAME   : 0
MISSPELLED  : 2
FROM_FRIEND : 0
...
```

```
# free      : 0
YOUR_NAME   : 1
MISSPELLED  : 1
FROM_FRIEND : 1
...
```

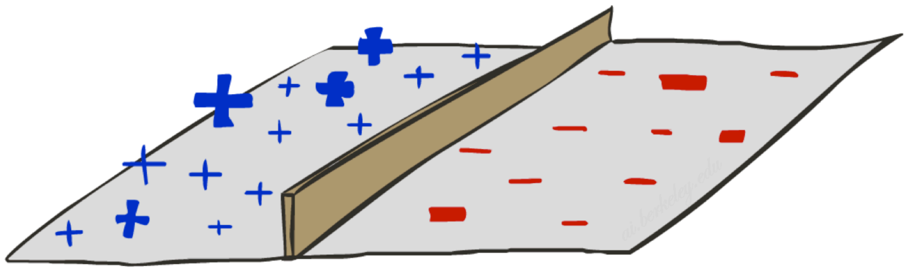
Dot produc $w \cdot f$ *positive means the positive class*

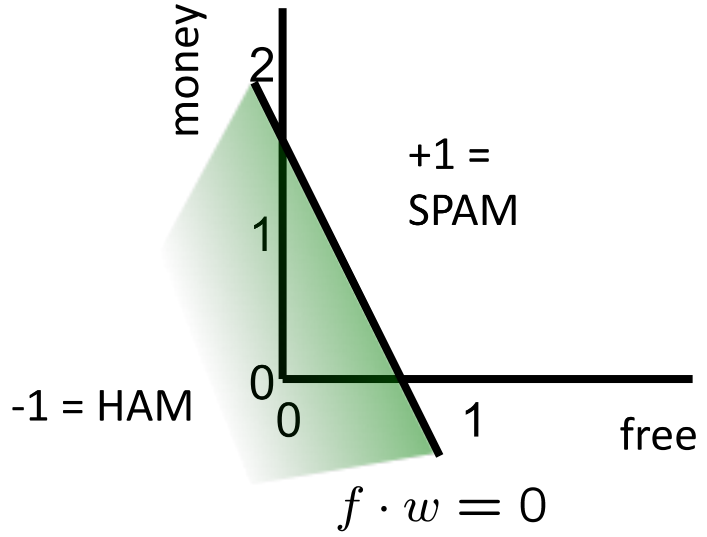
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - Decision boundary:
    - One side corresponds to Y=+1
    - Other corresponds to Y=-1

w

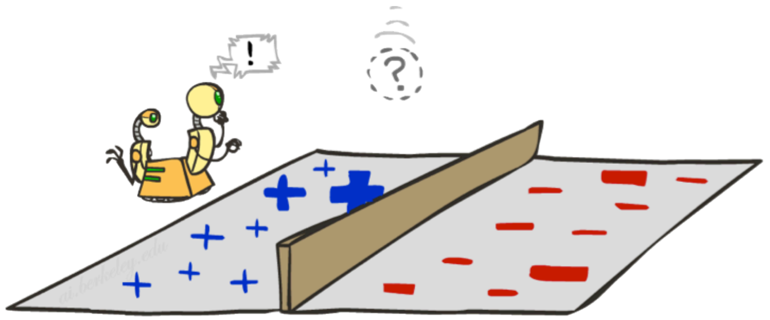
```
BIAS  : -
3
free  :
4
money :
2
...
```



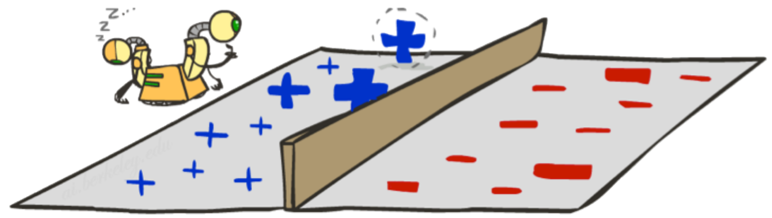


Learning: Binary Perceptron

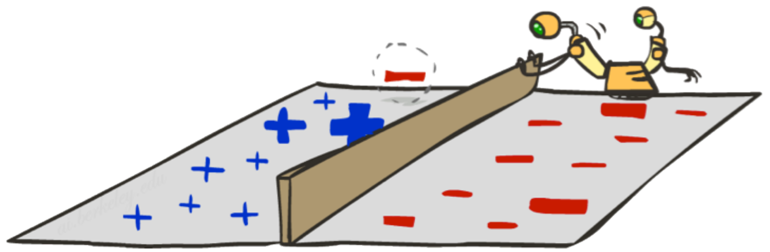
- Start with weights = 0
- For each training instance:
  - Classify with current weights



  - If correct (i.e., y=y*), no change!



  - If wrong: adjust the weight vector



Multiclass Decision Rule

- If we have multiple classes:
  - A weight vector for each class:

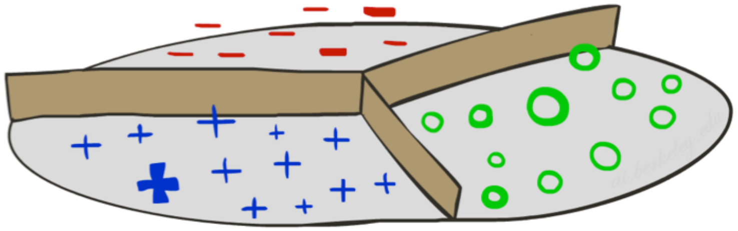
    $$w_y$$

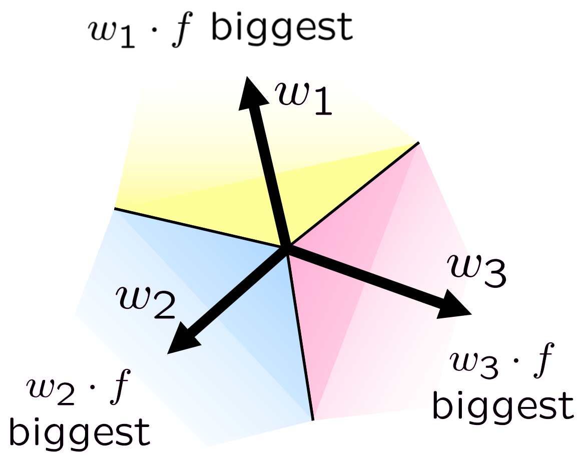
  - Score (activation) of a class y:

    $$w_y \cdot f(x)$$

  - Prediction highest score wins

    $$y = \arg\max_y \; w_y \cdot f(x)$$





Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

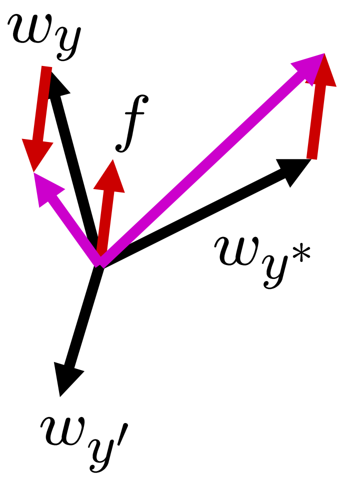
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y \ w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

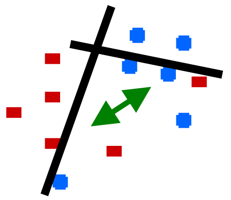
$$w_y = w_y - f(x)$$

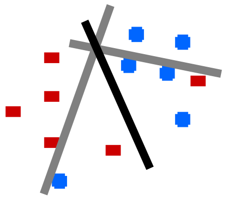
$$w_{y^*} = w_{y^*} + f(x)$$

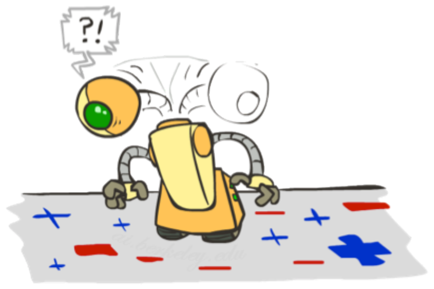


Problems with the Perceptron

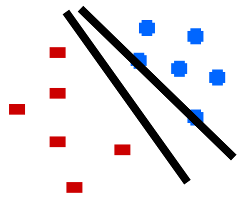
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



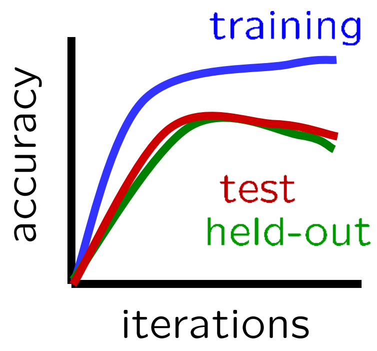


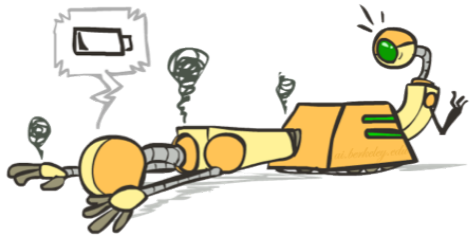


- Mediocre generalization: finds a "barely" separating solution



- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting





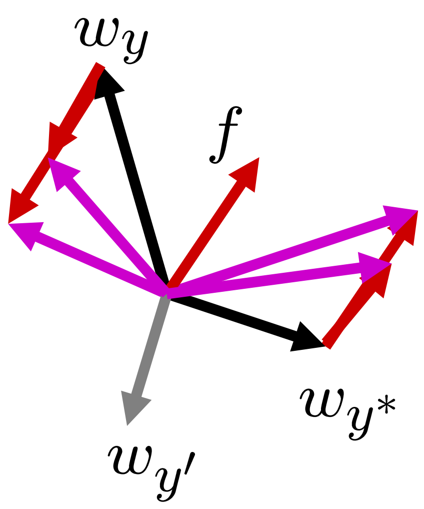
Fixing the Perceptron

- Idea: adjust the weight update to mitigate these effects
- MIRA*: choose an update size that fixes the current mistake…
- … but, minimizes the change to w

$$\min_w \frac{1}{2}\sum_y ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

- The +1 helps to generalize



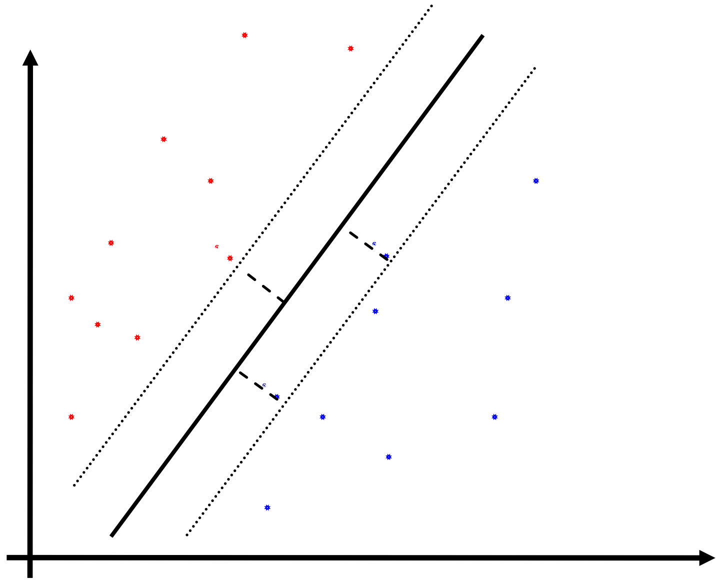
Guessed y instead of y^* on example x with features $f(x)$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$

* Margin Infused Relaxed Algorithm

Support Vector Machines

- Maximizing the margin: good according to intuition, theory, practice
- Only support vectors matter; other training examples are ignorable
- Support vector machines (SVMs) find the separator with max margin
- Basically, SVMs are MIRA where you optimize over all examples at once



MIRA

$$\min_w \frac{1}{2}||w - w'||^2$$

$$w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

SVM

$$\min_w \frac{1}{2}||w||^2$$

$$\forall i, y \; w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

Problem set question

Now you first decide to use a perceptron to classify your data. This problem will use the multi-class formulation even though there are only two classes. Suppose you directly use the scores given as features, together with a bias feature. That is f0 = 1, f1 = score given by A and f2 = score given by B. You want to train the perceptron on the training data in the table below.

Movie Name	A	B	*Profit?*
Meet Pac Man	3	2	Yes
Pixels	1	1	No
The Ghostly Adventures	4	5	No
Pac Baby	2	4	Yes
Pac is Back	3	4	Yes

Profit	Weights	Weights after 1st update
Yes	$[-1, 0, 0]$	
No	$[+1, 0, 0]$	

- Which is the first training instance at which you update your weights? Why?
- Write the updated weights after the first update.